UNREAL ENGINE

# Unreal Engine 4 学习总动员

## 蓝图应用

张宝荣◎编著

VISIT…

LANZAROTE Caliente.com

# Unreal Engine 4 学习总动员

## 蓝图应用

张宝荣◎编著

中国铁道出版社有限公司
CHINA RAILWAY PUBLISHING HOUSE CO., LTD.

# 版权信息

书名：Unreal Engine 4学习总动员——蓝图应用
作者： 张宝荣
排版：中国铁道出版社
出版社：中国铁道出版社
出版时间：2019.07
ISBN：978-7-113-25778-1

— · 版权所有 侵权必究 · —

# 内容简介

蓝图是Unreal Engine 4的重大特色，将程序可视化操作，可使复杂枯燥的代码编程变得极具逻辑性思维，使用户不用写一行代码就可以完成所有的工作。本书通过对几十个实战案例教程的操作演示，使复杂的开发工作变得极其有趣。具体内容包括：常规脚本、建造蓝图、蓝图通信、蓝图编辑器、蓝图技术指南、蓝图变量等。

配套资源赠送超过2小时的视频教学，精选了10个案例教程开展教学，手把手教会读者真正理解和使用Unreal Engine 4的核心精华——蓝图操作。

本书适合Unreal Engine初级用户阅读学习，可作为游戏开发、虚拟现实开发相关行业从业人员的参考书，也可作为大中专院校和社会培训机构相关专业的教材。

# 前言

配套资源下载地址：

http://www.m.crphdm.com/2019/0515/14084.shtml

放眼全球，纵观当今的时代，数字化、信息化、网络化是我们人类发展不可逆转的趋势。随着下一代互联网IPV6以及5G通信标准的逐步应用，人们的工作、生活、娱乐等领域都将发生革命性的变化。这其中以“虚拟现实”“人工智能”“大数据应用”“数据安全”等领域最为突出。在可以预见的未来，上述4个领域将引领世界科技发展潮流。

Epic Games公司成立于1991年，公司总部位于美国北卡罗来纳州卡里镇，在美国、欧洲、日本、中国和韩国等国家和地区设有工作室。Epic Games的作品包括《Unreal》（虚幻系列游戏）、《Gears of War》（战争机器）、《Infinity Blade》（无尽之剑）、《Paragon》（虚幻争霸）、《Fortnite》（堡垒之夜）、《SPYJiNX》（特工金克斯）、《BattleBreakers》（战争破坏者）、《Robo Recall》（机械重装），以及新的《Unreal Tournament》（虚幻竞技场）。1998年随着

《Unreal》（虚幻系列游戏）的推出，公司随即将开发这款游戏的工具也一并推出，供全球的游戏制作玩家免费使用，由此标志着Unreal Engine的正式诞生。

2014年Epic Games公司推出了Unreal Engine 4（虚幻引擎4，简称UE4，本书在不作特别说明时，都简称为UE4）版本，并且将其源代码全部公开。UE4进行了全新的渲染引擎升级，从而大大提升了渲染质量和速度。

UE4是一套为使用实时技术的人士开发的完整开发工具。从企业应用和电影体验到高品质的PC、主机、移动、VR及AR游戏，UE4都能为用户提供从启动项目到发行产品所需的一切，在同类产品中独树一帜。UE4提供了强大的工具套件以及简易的工作流程，能够帮助开发者快速迭代概念并立即查看成品效果，且无须触碰一行代码。而完整公开的源代码则能让UE4社区的所有成员都能够自由修改和扩展引擎功能。

UE4官方发布了许多的视频教程和在线帮助文档，以供用户学习和使用。另外，还公布了大量的游戏制作项目工程，免费供全球用户使用。由于各种原因，国内目前关于UE4方面的学习资源极为稀少。鉴于此，十分有必要推出一套全面介绍UE4技术内容的丛书，以供国内用户学习和使用。

本系列图书全面介绍了UE4的全部内容。丛书共有6本，分为快速入门、材质渲染、蓝图应用、动画设计、游戏开发、C++编程。内容包含了UE4的全部模块内容。本套图书具有鲜明的特色，首先，整套图书以案例教程为核心，每本书有数十个案例教程。手把手教会你快速上手UE4，使学习UE4变得极为容易，完全以实战操作为成书标准。其次，整套图书配有近18小时的语音视频教程，完全是精典案例实战操作式教学。最后，本套图书配有巨量的工程数据文件，以供读者非常方便地调用和查看。

由于编写这套图书工作量巨大，加之UE4更新较快，书中难免有不足和谬误之处，欢迎广大读者批评斧正。该套图书在开发过程中得

到了Epic Games公司和许多业内人士的大力支持和帮助，在此特别表示感谢。

作者
2019年6月

配套资源下载地址：
http://www.m.crphdm.com/2019/0515/14084.shtml

技术支持QQ群：596664789

第1章

# 蓝图简介

UE4中的蓝图——可视化脚本系统是一个完整的游戏脚本系统，其理念是，在虚幻编辑器中，使用基于节点的界面创建游戏可玩性元素。和其他一些常见的脚本语言一样，蓝图的用法也是通过定义在引擎中的面向对象的类或者对象。在使用UE4的过程中，常常会遇到在蓝图中定义的对象，并且这类对象常常也会被直接称为“蓝图（Blueprint）”。

该系统非常灵活且非常强大，因为它为设计人员提供了一般仅供程序员使用的所有概念及工具。另外，在UE4的C++实现上也为程序员提供了用于蓝图功能的语法标记，通过这些标记，程序员能够很方便地创建一个基础系统，并交给策划进一步在蓝图中对这样的系统进行扩展。

## 1.1

## 蓝图的工作原理

从蓝图的基本形式上讲，蓝图是针对你游戏添加的可视化脚本。通过使用连线把节点、事件、函数及变量连接到一起，它可以创建复杂的游戏性元素。

蓝图通过各种用途的节点构成图表来进行工作，这些节点包括针对蓝图每个实例的对象构建、独立的函数、一般的游戏性事件，从而实现各种行为及其他功能。

1.2

## 常用的蓝图类型

你将应用的最常见的蓝图类型是Level Blueprints（关卡蓝图）和Class Blueprints（类蓝图）。

这仅是其中的两种蓝图类型，还有蓝图宏和蓝图接口。

如图1-1所示为通过命令打开关卡蓝图。

图1-1

关卡蓝图的作用和UE3中的Kismet的作用一样，并且具有同样的功能。每个关卡具有其自己的关卡蓝图，它可以引用及操作关卡中的Actor，使用Matinee Actor控制过场动画，并管理类似于关卡动态载入、检查点及其他关卡相关的系统。关卡蓝图还可以和放置在关卡中的类蓝图（请参照下面的部分获得相关示例）进行交互，比如读取或设置任何变量、触发它们可能包含的自定义事件。

蓝图类，如图1-2所示。

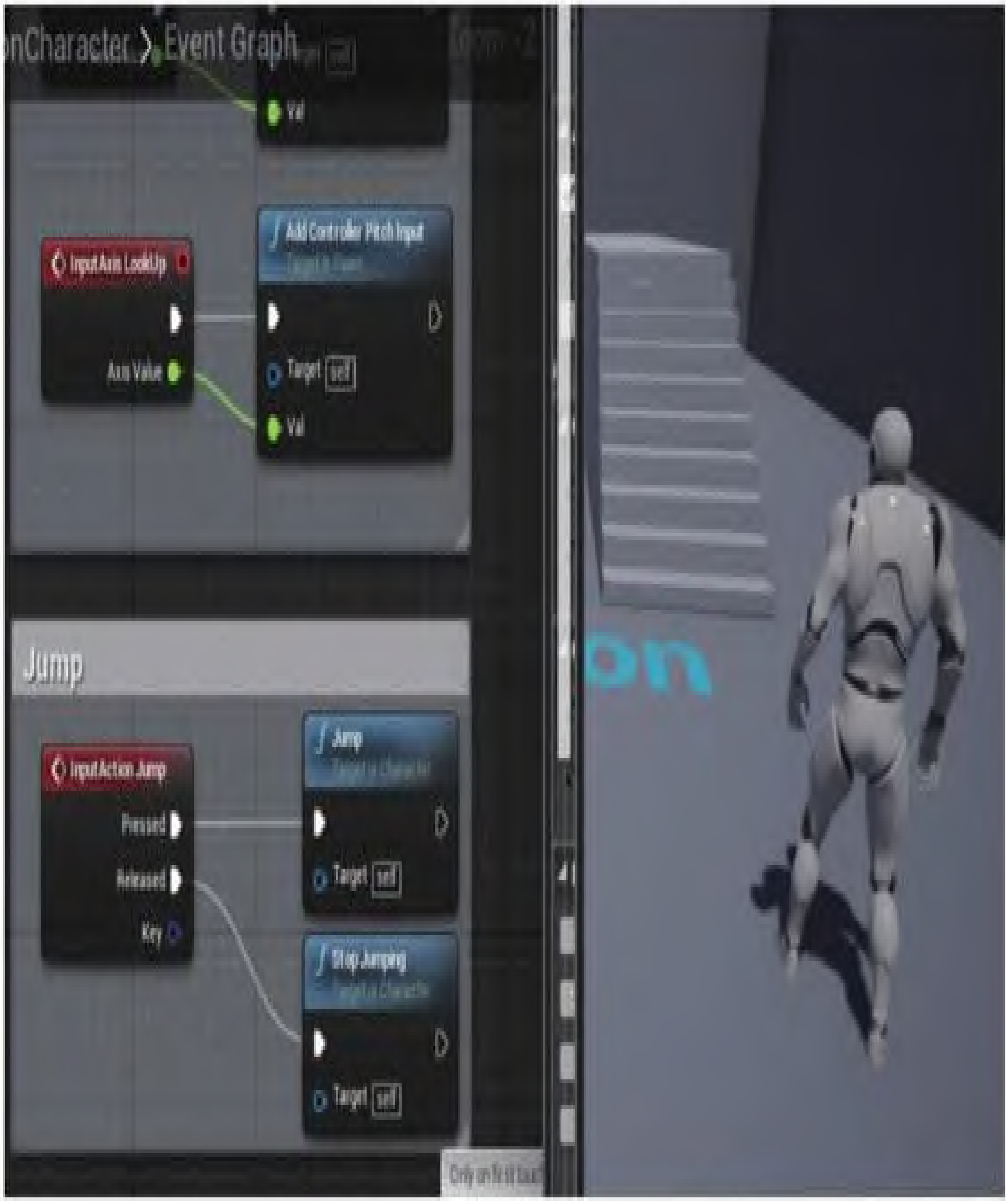



图1-2

蓝图类非常适合于制作交互式的资源，比如门、开关、可收集的道具及可破坏的景观。在上面的图片中，按钮和每个门都是独立的蓝图，包含了对玩家重叠事件做出反应的必要脚本，使得它们可以产生动画、播放音效、改变材质（比如，当按下按钮时变亮）。

在该示例中，按下按钮激活了门蓝图中的一个事件，使得门打开，但是门既可以通过其他类型的蓝图激活，也可以通过关卡蓝图序列激活。由于蓝图的自包含特性，所以你可以通过这种方式构建蓝图，把它们放置到关卡中，它们就可以正常运作了，仅需要少量设置。这也意味着编辑整个项目时使用的一个蓝图将更新蓝图的每个实例。

1.3

## 蓝图还能够做什么

你已经阅读了关于关卡蓝图和类蓝图的信息，下面列出了几个可以通过蓝图系统完成的实例。

使用构建脚本创建自定义的预制，如图1-3所示。

图1-3

Construction Script（构建脚本）是类蓝图中的一种图表类型，当在编辑器中放置或更新Actor时会执行它，但是在游戏过程中不会执行它。这对于轻松地创建自定义的道具是非常有用的，使得环境美工人员工作效率更高，比如，一个光源设备，它可以自动地更新材质来匹配点光源组件的颜色和亮度；或者一个蓝图，可以在一个区域上随机地散布植被网格物体。

在Content Examples（内容示例）地图中，包含多个示例的很长的房间（如上面的图片所示）实际上是由多个组件构成的一个蓝图。蓝图的构建脚本根据蓝图的详细信息面板中暴露的参数创建及排列各种静态网格物体及光源。对于我们创建的每个Content Example（内容示例）地图，我们都可以把那个示例房间蓝图放置在其中，设置要生成的房间的长、高及数量（及其他几个选项），并且片刻就可以完成整套房间。

像这样的蓝图，初次创建时会比较耗时，但是如果你明白你要经常使用它，那么构建关卡时所节约的时间、修改时的方便程度，使得这样设计变得非常值得。

创建一个可运行的游戏角色，如图1-4所示。

图1-4

Pawn也是一种类型的类蓝图，你可以把你的可运行角色需要的每个元素放到蓝图图表中。你可以操作相机行为，设置鼠标、控制器及触屏的输入事

件，以及创建动画蓝图资源来处理骨架网格物体动画。

当你创建一个新的角色蓝图时，它自带一个角色组件，具备了到处移动、跳跃、游泳、下降等所需的大部分行为，你所需要做的就是根据你想控制角色的方式来添加一些输入事件。

创建HUD，如图1-5所示。

蓝图脚本也可以用于创建游戏的HUD，这和类蓝图类似，因为它可以包含事件序列和变量，但是它会被分配给你的项目的GameMode（游戏模式）资源，而不是直接添加到关卡中。

你可以设置一个HUD来从其他蓝图中读取变量，并使用这些变量来显示一个生命值条、更新分数、显示目标标记等。你还可以使用HUD给类似于可以单击的按钮或手游中可以对触摸输入做出反应的按钮添加命中区域。

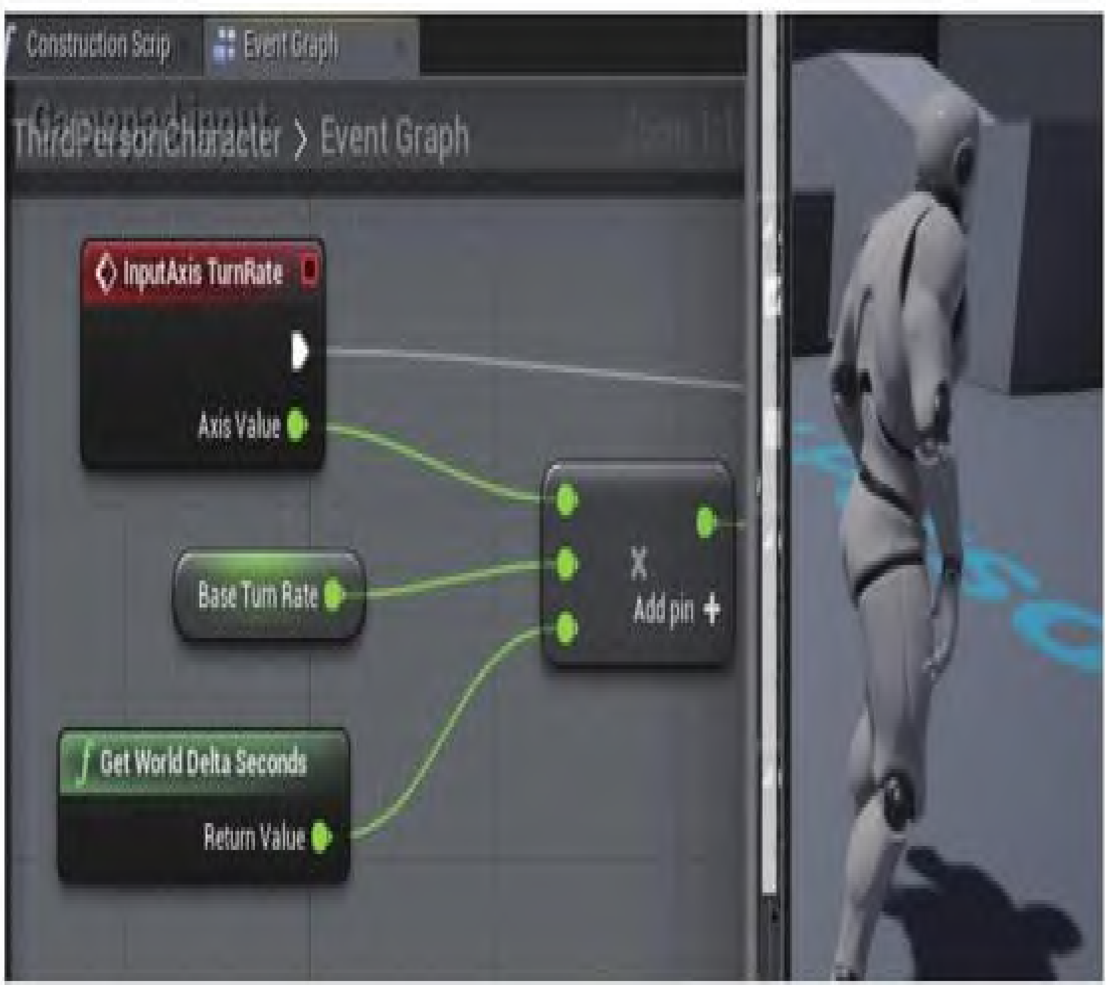



图1-5

上面图片中的所有示例都存在于UE4的示例内容中，所以如果你想仔细查看相关内容，你可以在Content Examples、Shooter Game和Swing Ninja中找到它。

1.4

## 蓝图编辑器和图表

无论你正在构建一个关卡蓝图还是一个类蓝图，你将会用到蓝图编辑器参考中的蓝图元素，如图1-6所示。

File Edit Asset View Debug Window Help
Components
Add Component
Compile
Save
Browse
Find
Class Settings
Simulation
Play
Details
Viewport
Construction Script
Event Graph
ThirdPersonCharacter › Event Graph
Movement input
Jump
Touch input
BLUEPRINT
My Blueprint
Add New
Graphs
EventGraph
Functions
ConstructionScript
Macros
Variables
My Character
BaseTurnRate
BaseLookUpRate
Event Dispatchers
Default
Actor Tick
Character
Camera
Pawn
Rendering

图1-6

根据你所应用的蓝图类型的不同，有各种不同类型的蓝图编辑器。大部分蓝图编辑器的核心功能都是图表模式，通过使用其核心的图表选卡可以布局你的蓝图网络。

第2章

# 常规脚本

2.1

# 基础脚本

蓝图为脚本语言提供了可视方案。因此，该系统拥有标准脚本编写语言的许多细微差别，如数据类型变量、阵列、结构体等。虽然蓝图要求每个节点的显式线性执行，执行流程仍然按照典型脚本语言中的方式进行。下文将详细说明不同变量类型、这些变量的使用，以及图表中节点的执行。

## 2.1.1　变量

变量可创建为各种不同类型，如布尔、整数、浮点数据类型。它们拥有颜色编码，便于在蓝图中进行识别。蓝图变量也可以是保存对象、Actor和类等资源的引用类型。

## 2.1.2　执行流程

在蓝图中，首先执行的是事件节点，然后执行流程跟随白色执行引线从左至右进行。游戏在编辑器中运行时可显示执行流程，有助于进行调试。数据通过带颜色的引线流动，匹配相应的变量类型。节点执行时将对输入引脚进行求值，从右至左反推数据引线，直到算出最终结果并供应到节点。

带执行引脚的节点（非纯节点）在执行时保存其输出引脚的数值，而不带执行引脚的节点（纯节点）在连接到其输出的节点每次执行时对输出进行重新求值。

### 2.1.3 阵列和结构

阵列和结构是多种数据类型的合集。它们可用于更复杂的系统，有助于集中相关信息，并保证大量信息的流线型处理。

2.2

## 蓝图中的转换

在蓝图中使用Cast To节点时，简单而言，就是在尝试检查发出转换的对象是否为被转换的特定对象。换言之，你创建了一个特殊的角色蓝图（如名为MyCharacter），其中包含变量或其他自定义功能，并将其作为默认Pawn类（或所有玩家角色默认使用的角色蓝图）指定。

如需从另一个蓝图访问玩家角色的属性，可使用Get Player Character节点以常规方式（设置/获取其位置、旋转灯）对玩家角色产生影响，但无法访问已添加到MyCharacter蓝图的自定义功能，因为你获取的是玩家角色而非特定类型的角色。

使用Get Player Character节点，再使用Cast To MyCharacter节点（特殊的角色蓝图），即可确认玩家角色是否为MyCharacter，以便访问该蓝图中所包含的变量、函数、事件或其他特殊功能。

在此节中你可了解到转换的部分使用情况、查看Cast To节点的使用范例、了解关于目标蓝图转换的内容以及其他类型的转换。

以下是几个蓝图转换使用范例。

•需要访问另一个蓝图的特殊版本。

◊角色走进火焰中，导致体力值耗尽。

▪投射到特殊的角色蓝图，以便访问并变更体力值。

◊角色死亡，需要重新生成。

▪ 投射到特殊的游戏模式蓝图，执行重新生成脚本。

•需要访问相同类的多个蓝图，并以相同方法进行修改。

◊场景中拥有数盏灯，事件发生时需将它们开启或关闭。

▪ 投射到灯蓝图并执行函数将灯关闭。

•需要访问一个特殊的子蓝图。

◊存在基于一个动物蓝图（猫、狗、鸟）的数个蓝图，需要访问其中一个动物。

▪ 投射到猫、狗和鸟，访问其相应的蓝图和特有功能。

## 2.3

# 事件

事件（Events）是从游戏性代码中调用的节点，在事件图表（EventGraph）中开始执行个体网络。它们使蓝图执行一系列操作，对游戏中发生的特定事件（如游戏开始、关卡重置、受到伤害等）进行回应。这些事件可在蓝图中访问，以便实现新功能，或覆盖/扩充默认功能。任意数量的Events均可在单一EventGraph中使用，但每种类型只能使用一个，如图2-1所示。

图2-1

每个事件只能执行单个对象。如需从一个事件触发多项操作，需要将它们线性地排列起来。

在使用Matinee修改蓝图属性时可查阅使用Matinee创建事件的内容，可在过场序列播放中的特定时间触发这些事件。

### 2.3.1 Event Level Reset

此蓝图事件节点仅在关卡蓝图中可用。

此蓝图事件节点仅在服务器上执行。在单人游戏中，本地客户端即视为服务器，如图2-2所示。

Level Reset事件在关卡重启时发出执行信号。它在关卡重新加载后进行某项触发时非常实用。如玩家角色已死亡，但关卡无需重新加载时，如图2-3所示。

图2-2

图2-3

## 2.3.2 Event Actor Begin Overlap

多项条件同时满足时，将执行该事件。

•Actor之间的碰撞响应必须允许重叠。

•执行事件的两个Actor的Generate Overlap Events均设为true。

•最后，两个Actor的碰撞开始重叠；两者移到一起或其中一个创建时与另一个重叠，如图2-4所示。

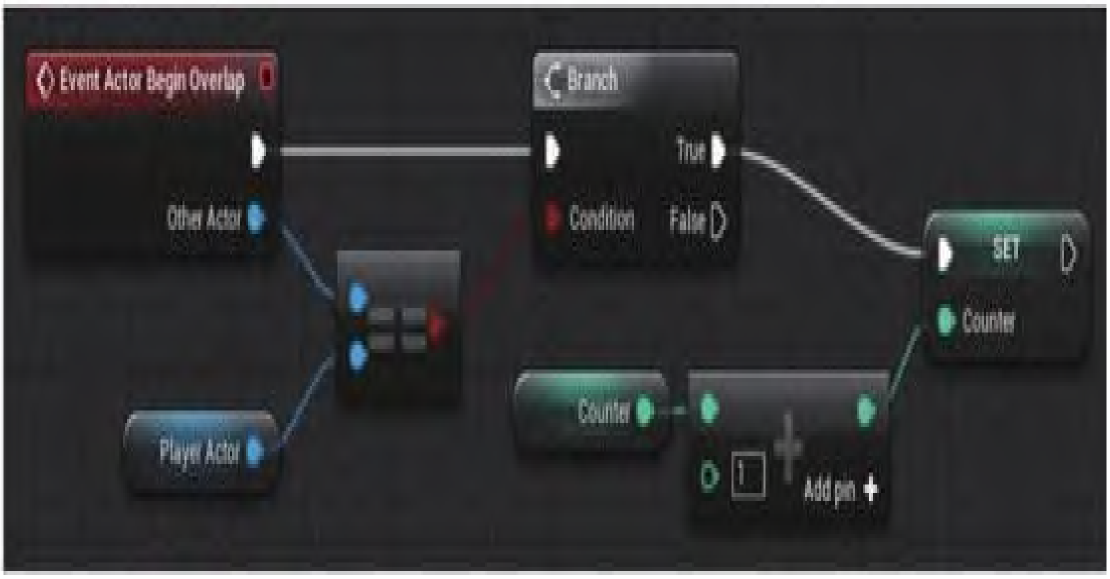



图2-4

此蓝图Actor和保存在Player Actor变量中的Actor重叠时，它将增加Counter整数变量。

| 项　目 | 描　述 |
| --- | --- |
| 输出引脚 | |
| Other Actor | Actor—与此蓝图发生重叠的 Actor |

## 2.3.3　Event Actor End Overlap

多项条件同时满足时，将执行该事件。

•Actor之间的碰撞响应必须允许重叠。

•执行事件的两个Actor的Generate Overlap Events均设为true。

•最后，两个Actor的碰撞停止重叠；它们将分离，或在其中一个将被销毁，如图2-5所示。

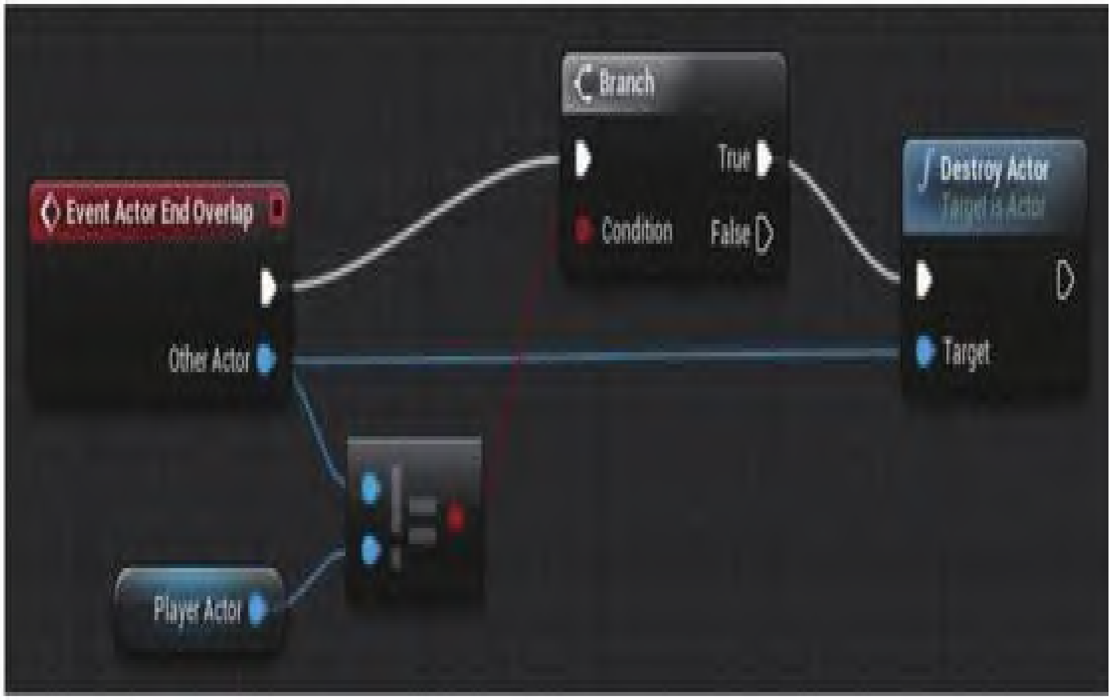



图2-5

当此蓝图Actor不与其他Actor发生重叠时（保存在Player Actor变量中的Actor除外），它将销毁重叠的Actor。

| 项　目 | 描　述 |
| --- | --- |
| 输出引脚 | |
| Other Actor | Actor——与此蓝图发生重叠的 Actor |

## 2.3.4　Event Hit

只要其中一个相关Actor的碰撞设置中Simulation Generates Hit Events设为true，该事件便会执行。

如你使用Sweeps创建运动，即使未选中标记也将获得此事件。只要Sweep阻止你移过阻挡物体，这便会发生。

| 项　目 | 描　述 |
|---|---|
| 输出引脚 | |
| My Comp | 原始组件——被命中的执行 Actor 上的组件 |
| Other | Actor——参与碰撞的其他 Actor |
| Other Comp | 原始组件——参与碰撞，被命中的其他 Actor 上的组件 |
| Self Moved | 布尔型——接受来自另一个物体运动的命中时使用（如为 false），对 Hit Normal 和 Hit Impact Normal 的方向进行调整，以便表现其他物体对被命中物体施加的力 |
| Hit Location | 矢量——两个碰撞 Actor 之间的接触位置 |
| Hit Normal | 矢量——碰撞方向 |
| Normal Impulse | 矢量——Actor 碰撞的力 |
| Hit | 命中结果结构体——一次命中中收集到的所有数据，可剥离并“打破”该结果，访问数据的单个位元 |

在此例中，蓝图执行Hit时，它将在冲撞点生成爆炸效果，如图2-6所示。

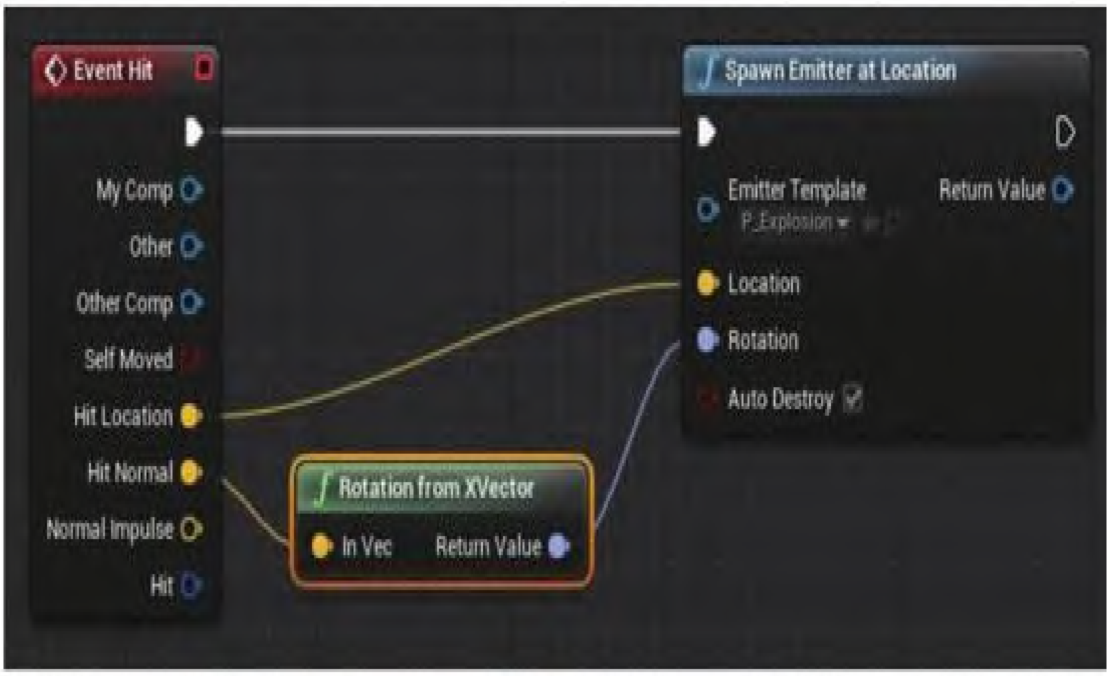



图2-6

## 2.3.5 Event Any Damage

此蓝图事件节点仅在服务器上执行。在单人游戏中，本地客户端即视为服务器，如图2-7所示。

此事件在造成整体伤害时出现。如溺死或环境伤害，并非点伤害或放射伤害。

图2-7

| 项　目 | 描　述 |
| --- | --- |
| 输出引脚 | |
| Damage | 浮点型——传入Actor的伤害量 |
| Damage Type | 伤害类型对象——这是在输出伤害上包含额外数据的对象 |
| Instigated By | Actor——负责伤害的Actor。这就是开枪或投手雷造成伤害的Actor |
| Damage Causer | Actor——输出伤害的Actor。这可以是子弹或爆炸 |

此处，如果对Actor造成的伤害来自水，将减少体力值并在屏幕上生成警告，如图2-8所示。

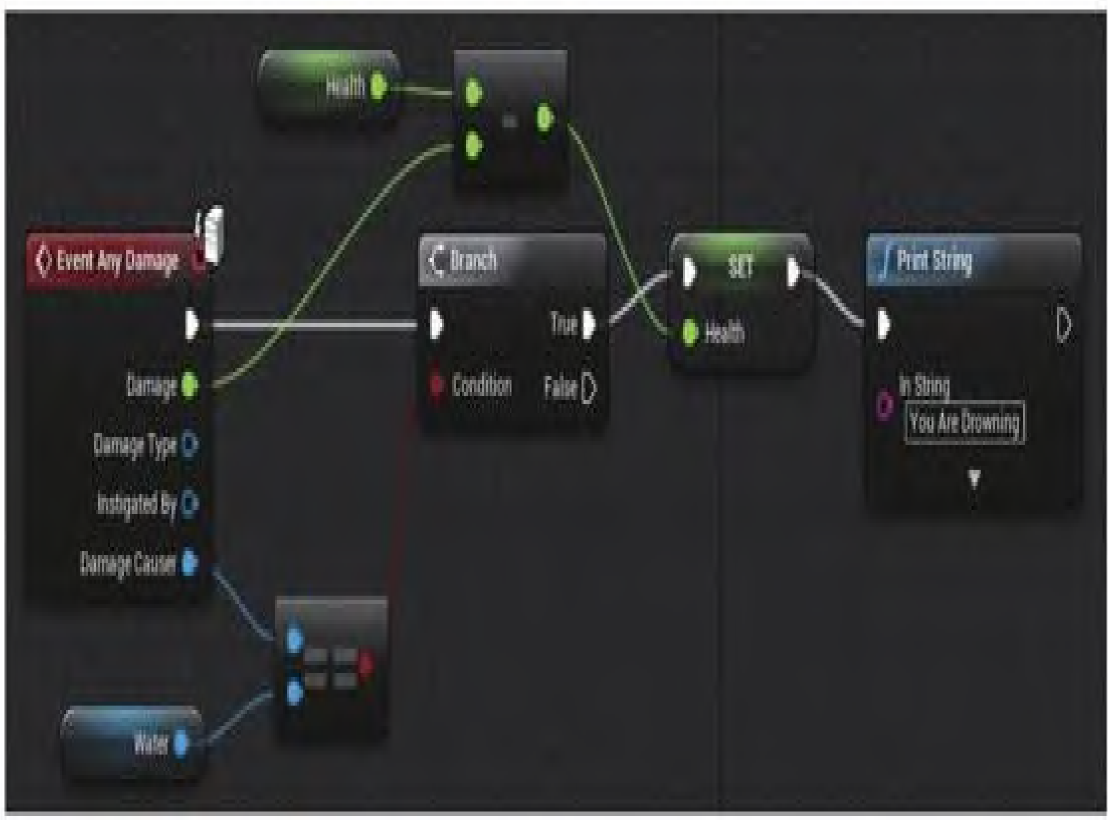



图2-8

## 2.3.6 Event Point Damage

此蓝图事件节点仅在服务器上执行。在单人游戏中，本地客户端即视为服务器，如图2-9所示。

点伤害（Point Damage）代表由投射物、扫射武器，甚至近战武器造成的伤害。

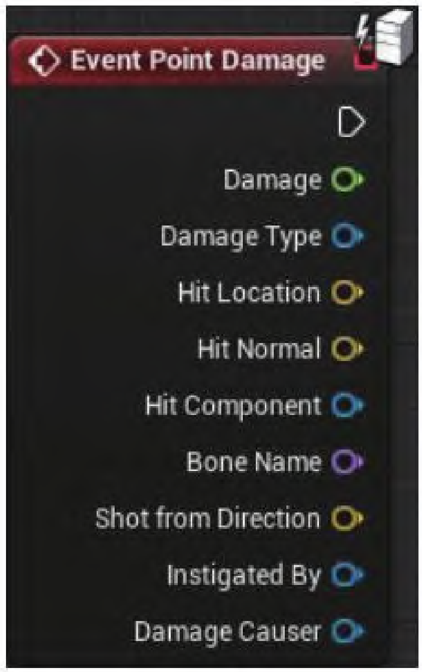



图2-9

| 项　目 | 描　述 |
| --- | --- |
| 输出引脚 | |
| Damage | 浮点型——传入Actor的伤害量 |
| Damage Type | 伤害类型对象——这是在输出伤害上包含额外数据的对象 |
| Hit Location | 矢量——应用伤害的位置 |
| Hit Normal | 矢量——碰撞方向 |
| Hit Component | 原始组件——被命中的执行Actor上的组件 |
| Bone Name | 名称——命中的骨骼名称 |
| Shot from Direction | 矢量——伤害来源的方向 |
| Instigated By | Actor——负责伤害的Actor。这是开枪或投手雷造成伤害的Actor |
| Damage Causer | Actor——输出伤害的Actor。这可以是子弹或爆炸 |

在此例中，受到任意伤害时均会从Actor的体力值减去造成的伤害。但如果Actor的头部被击中，则Actor的体力值设为-1，如图2-10所示。

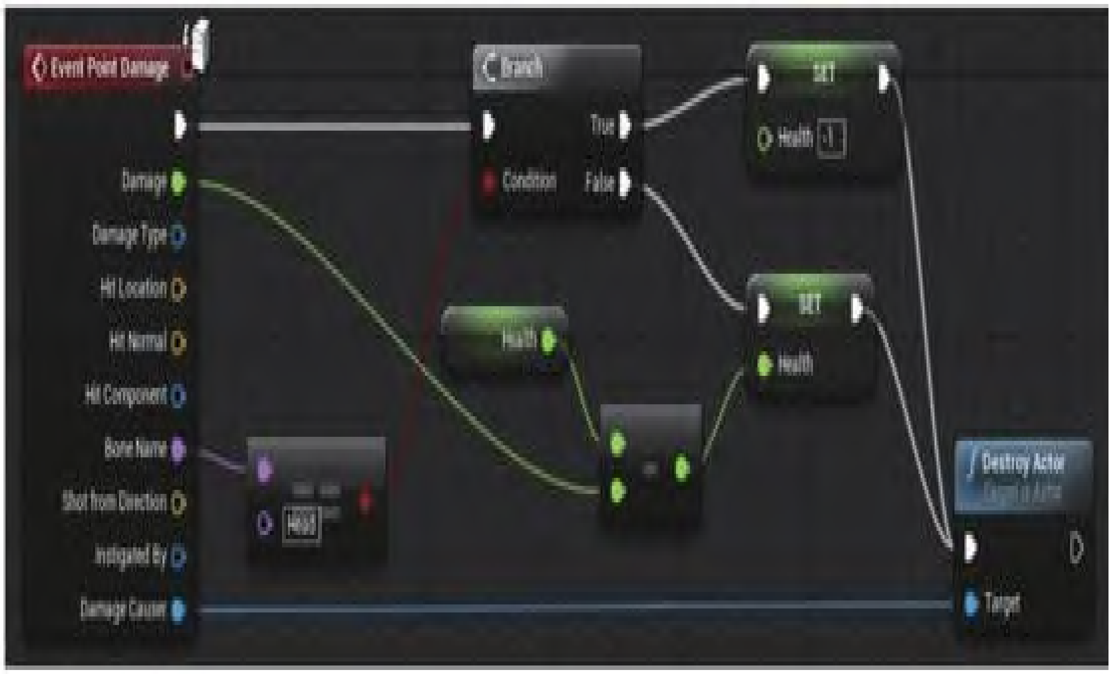



图2-10

## 2.3.7 Event Radial Damage

此蓝图事件节点仅在服务器上执行。在单人游戏中，本地客户端即视为服务器，如图2-11所示。

图2-11

放射伤害（Radial Damage）事件在该序列的父Actor受到放射伤害时调用。这可用于处理基于爆炸伤害或间接伤害的事件，如图2-12所示。

| 项　目 | 描　述 |
|---|---|
| 输出引脚 | |
| Damage Received | 浮点型——从事件接收的伤害量 |
| Damage Type | 伤害类型对象——这是在输出伤害上包含额外数据的对象 |
| Origin | 矢量——3D空间中的伤害来源位置 |
| Hit Info | 命中结果结构体——一次命中中收集到的所有数据，可剥离并“打破”该结果，访问数据的单个位元 |
| Instigated By | 控制器——发起伤害的控制器（AI或玩家） |
| Damage Causer | Actor——输出伤害的Actor，可以是子弹、火箭、激光或角色的拳击 |

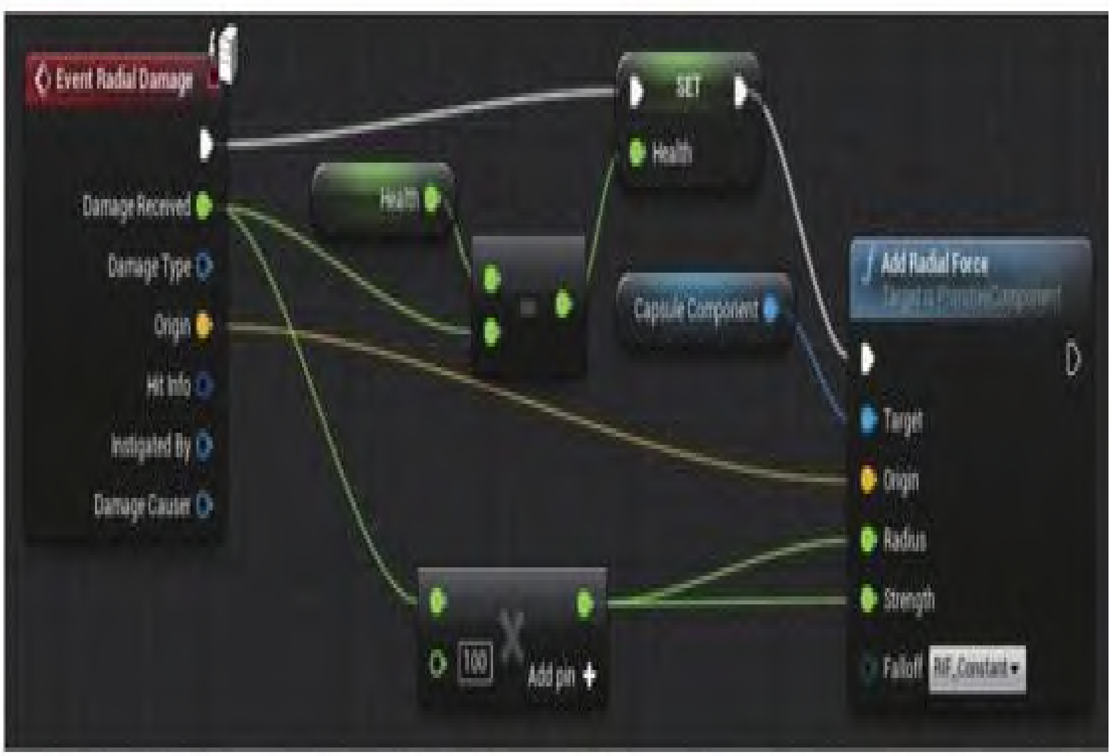



图2-12

### 2.3.8 Event Actor Begin Cursor Over

使用鼠标界面时，鼠标光标在Actor上悬停时执行的事件，如图2-13所示。

图2-13

鼠标经过此Actor后，它将把动态材质实例上名为Highlight的标量参数设为1.0。

### 2.3.9 Event Actor End Cursor Over

使用鼠标界面时，鼠标光标在Actor上移开时执行的事件，如图2-14所示。

图2-14

它将把保存在Target Notification中的Actor设为在游戏中隐藏。

## 2.3.10 Event Begin Play

游戏开始时将在所有Actor上触发此事件。游戏开始后生成的所有Actor上均会立即调用此事件，如图2-15所示。

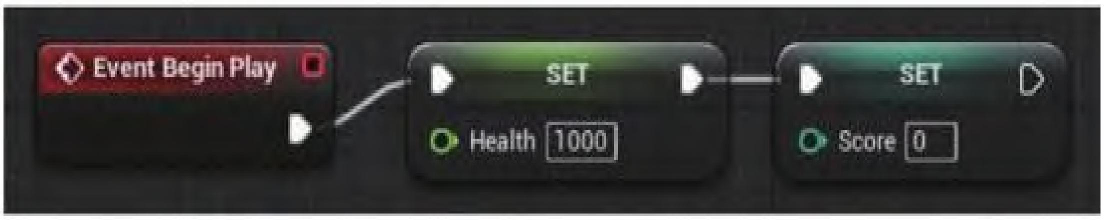



图2-15

开始游戏时，此Actor将把其体力值设为1000，分数设为0。

## 2.3.11 Event End Play

Actor不存在于世界场景中时执行此事件，如图2-16所示。

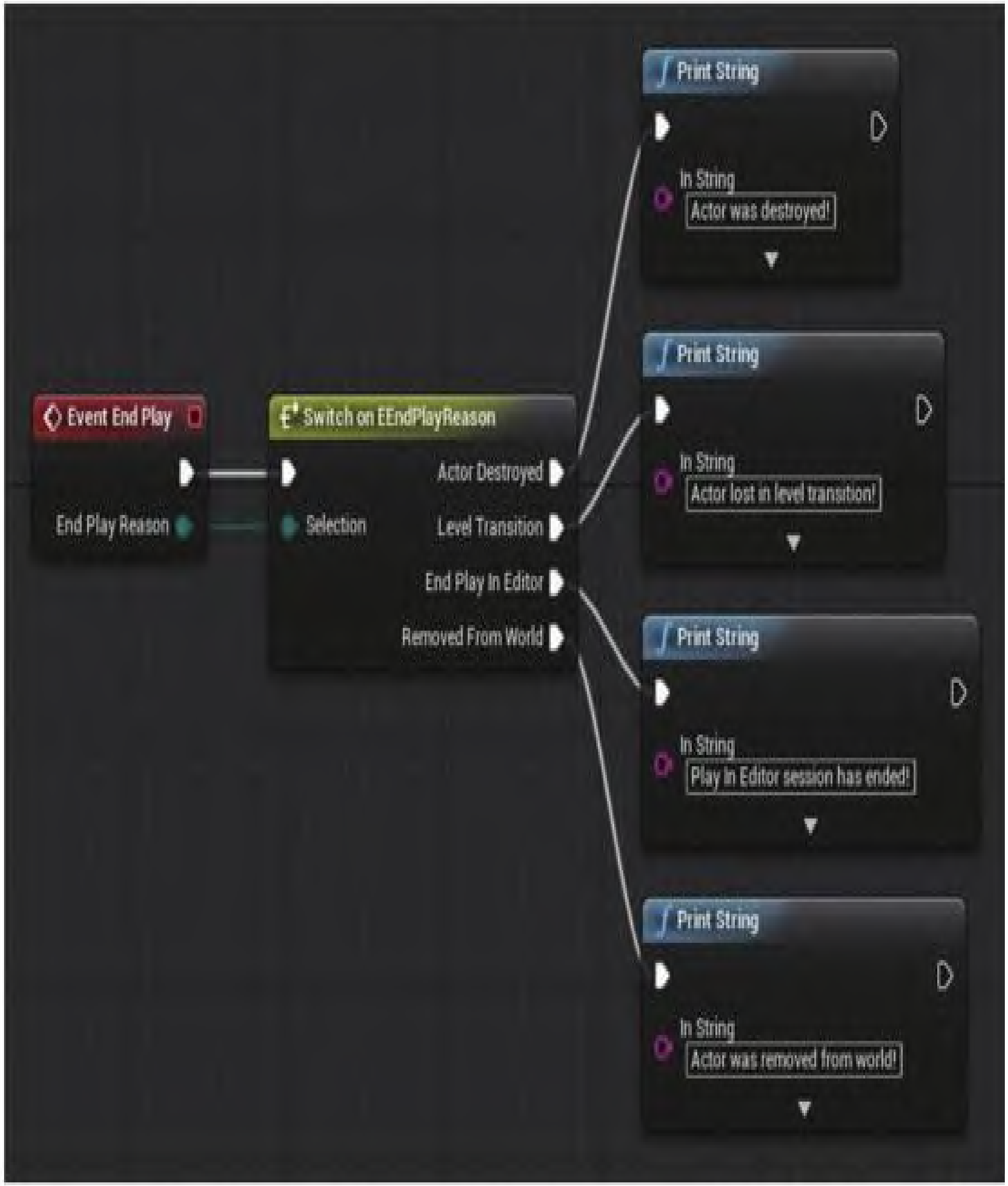



图2-16

此Actor不存在于世界场景中时，字符串将输出，说明事件被调用的原因。

| 项　目 | 描　述 |
| --- | --- |
| 输出引脚 | |
| End Play Reason | EEndPlayReason 枚举——说明 Event End Play 被调用原因的枚举 |

## 2.3.12 Event Destroyed

Actor被销毁时执行此事件，如图2-17所示。

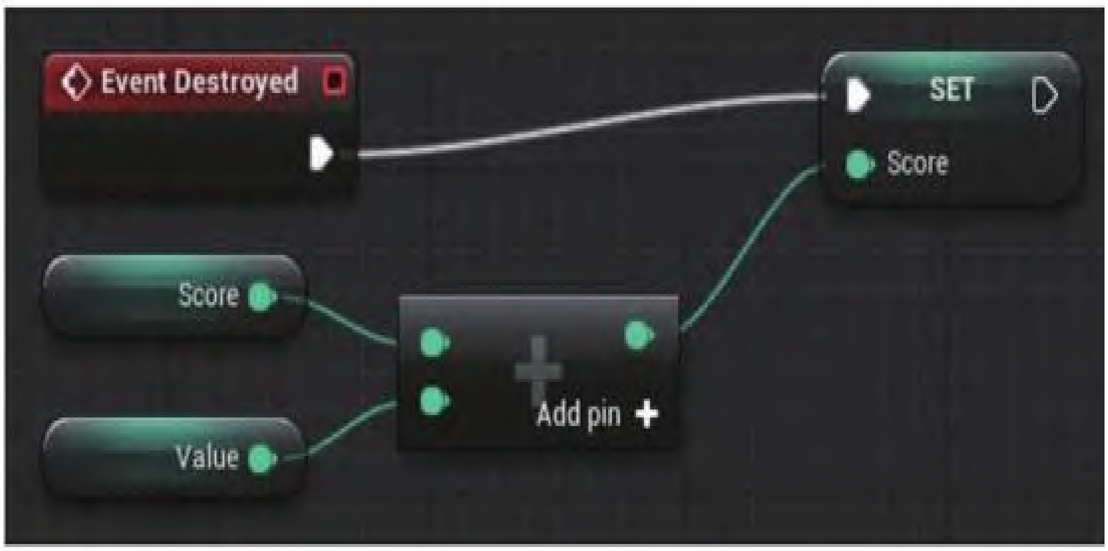



图2-17

此例中，Score变量设为Value加Score。

Destroyed事件将在之后的版本中移除！Destroyed函数的功能已合并到EndPlay函数。

## 2.3.13 Event Tick

游戏进程中每帧调用的简单事件，如图2-18所示。

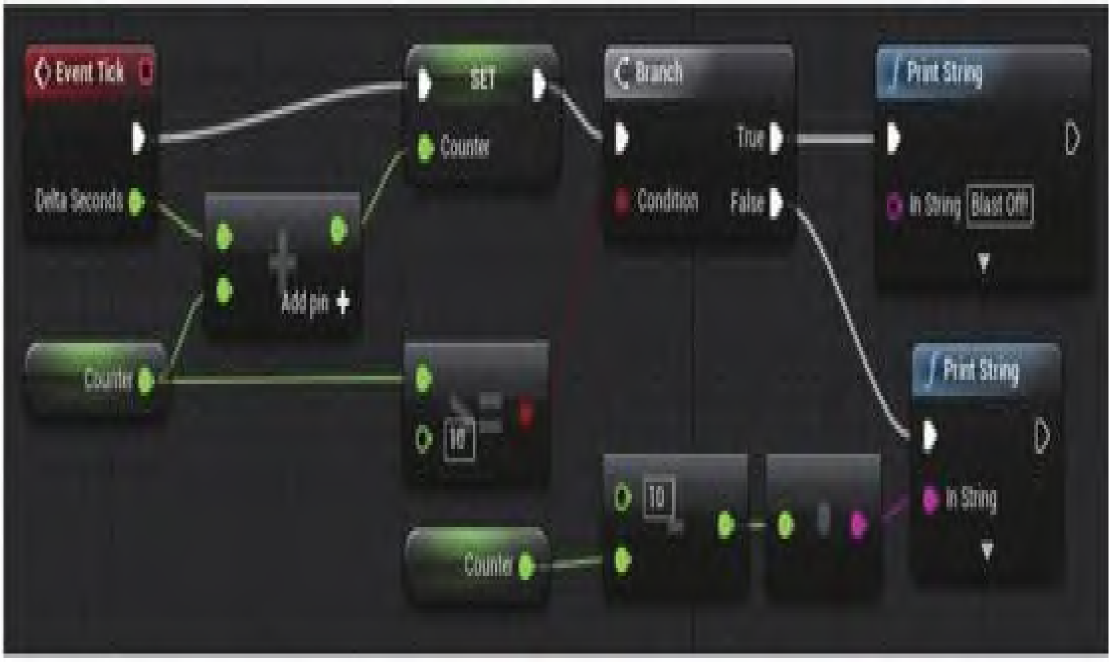



图2-18

此例使用Delta Seconds构成倒数计时器显示日志，最后的tick为“Blast Off！”

| 项　目 | 描　述 |
| --- | --- |
| 输出引脚 | |
| Delta Seconds | 浮点型——输出帧之间的时间量 |

## 2.3.14　Event Receive Draw HUD

此事件仅限继承自HUD类的蓝图类可用，如图2-19所示。

这是一个特殊事件，使蓝图可绘制到HUD。此事件须创建HUD绘制节点，如图2-20所示。

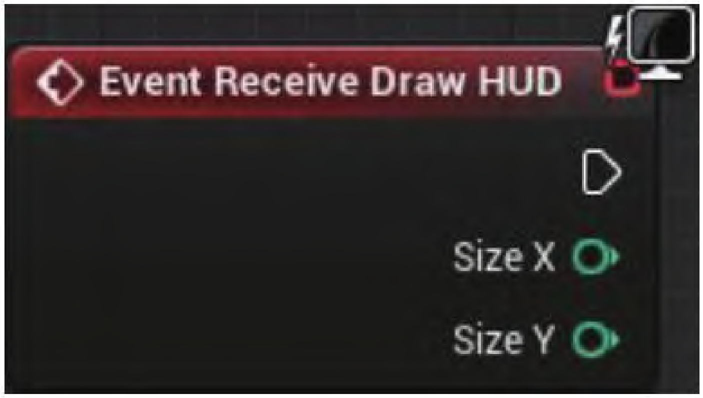



图2-19

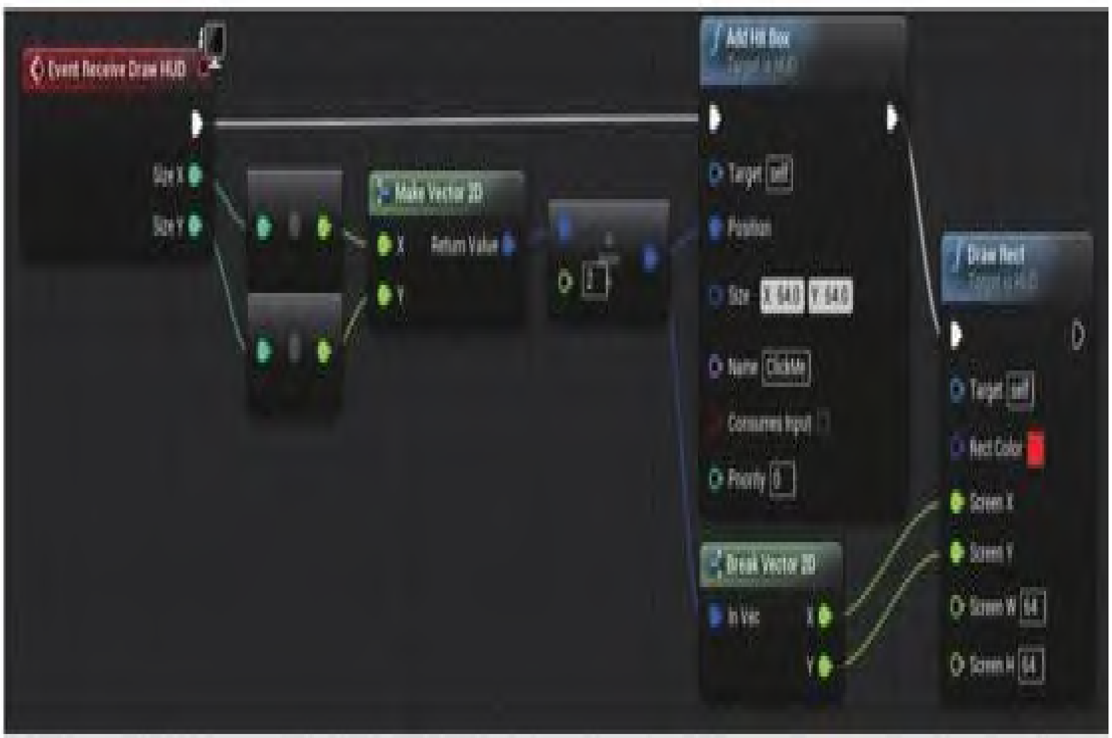



图2-20

这会在渲染窗口的中央创建一个可单击的Hit Box，其后有一个红框，便于辨认。

| 项　目 | 描　述 |
| --- | --- |
| 输出引脚 | |
| Size X | Int——渲染窗口的像素宽度 |
| Size Y | Int——渲染窗口的像素高度 |

## 2.3.15　Custom Event

自定义事件，如图2-21所示。

Cstom Events（自定义事件）为你提供了一种创建你自己的事件的方法，你可以在你的蓝图序列的任何地方调用这些事件。当你正在把多个输出执行线连接到一个特定节点的输入执行引脚时，使用自定义事件可以简化你图表的节点连线网络。你甚至可以在一个蓝图的图表中创建自定义事件，而在另一个图表中调用该事件。

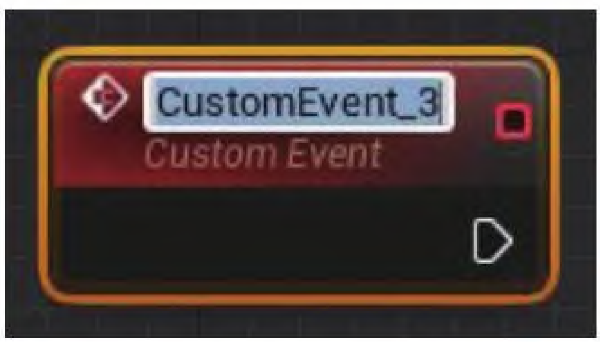



图2-21

创建自定义事件的简单流程。

•创建并命名Custom Event（自定义事件）。

•设置这个事件应该具有的任何输入参数及其默认值。

•创建一个调用该自定义事件的特殊函数。

•输入任何需要的输入参数。

### 2.3.15.1　创建自定义事件

**Step 1** 通过右击并从关联菜单中选择“Add Custom Event（添加自定义事件）”来创建一个Custom Event（自定义事件）节点，如图2-22所示。

**Step 2** 给这个新事件赋予一个名称，如图2-23所示。

图2-22

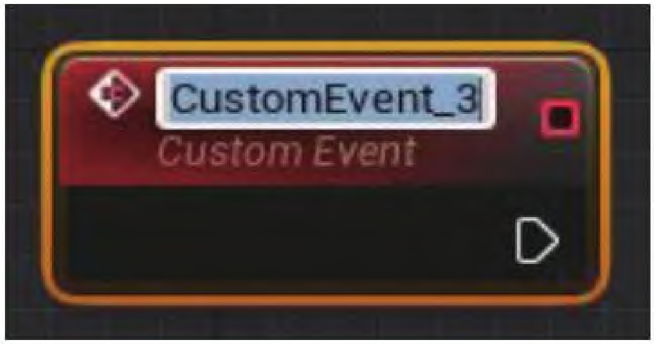



图2-23

**Step 3** 在你的新事件的Details（详细信息）面板中，你可以设置当在服务器上调用该事件并添加输入参数时，是否将该事件复制到所有客户端上，如图2-24所示。

图2-24

要想添加输入参数：

**Step 1**　单击详细信息面板的Inputs（输入）部分的New（新建）按钮，如图2-25所示。

图2-25

**Step 2**　命名该新输入并使用下拉菜单设置其类型。在这个示例中，有一个String（字符串型）的输入参数MyStringParam，如图2-26所示。

图2-26

Step 3　你也可以通过展开参数项来设置一个默认值，如图2-27所示。

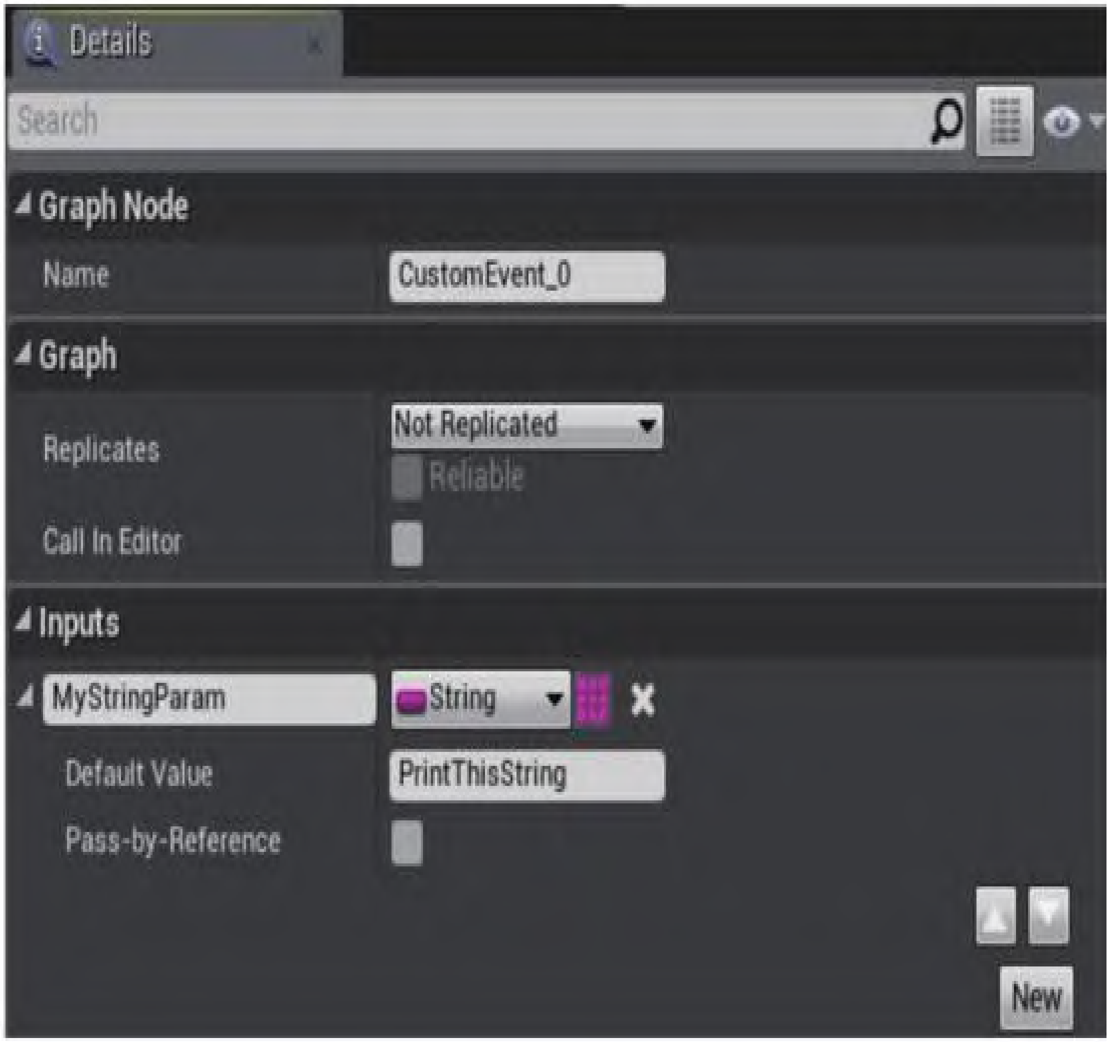



图2-27

要想改变节点边缘上的这个参数的引脚的位置，请使用展开的详细信息面板项的向下和向上箭头。

现在，和其他事件或执行节点一样，你可以把其他节点附加到你的自定义事件的输出执行引脚上，这样当触发了你的自定义事件时将会开始执行那个节点网络。这个自定义事件向屏幕输出了一个字符串，如图2-28所示。

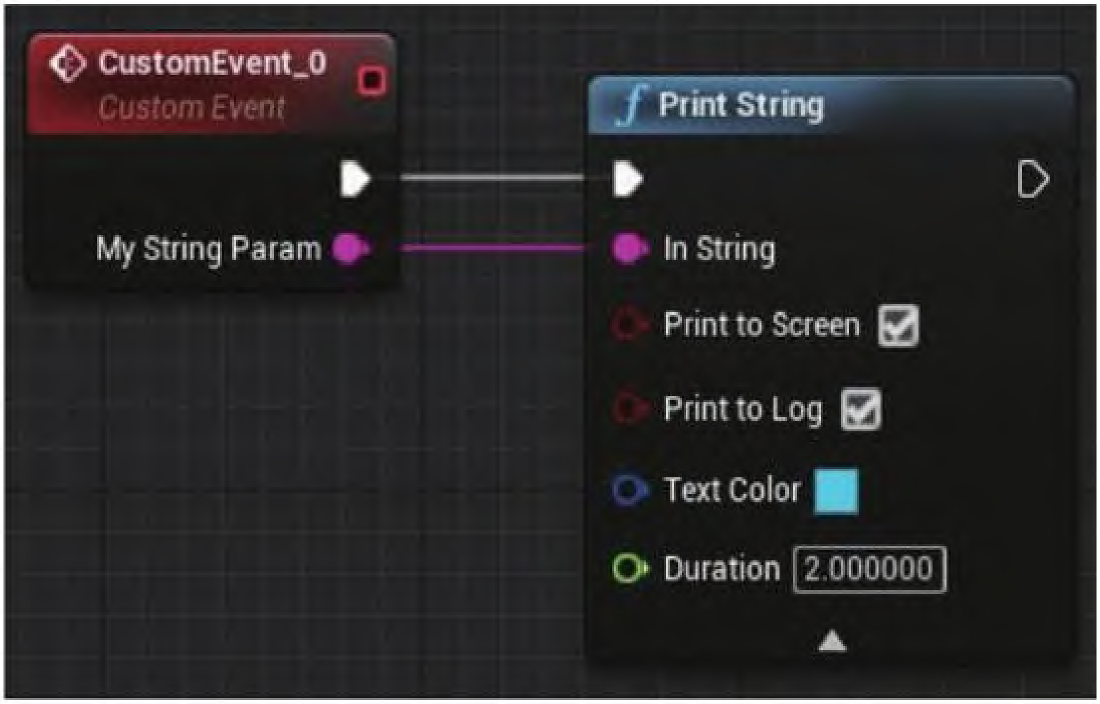



图2-28

#### 2.3.15.2 调用自定义事件

你已经创了自定义事件及其相关的节点网络，但是和常规的Events（事件）不同，没有触发自定义事件的预制条件。要想调用你的自定义事件，请右击并从关联菜单中选择Call Function（调用函数）>[自定义事件名称]，如图2-29和2-30所示。

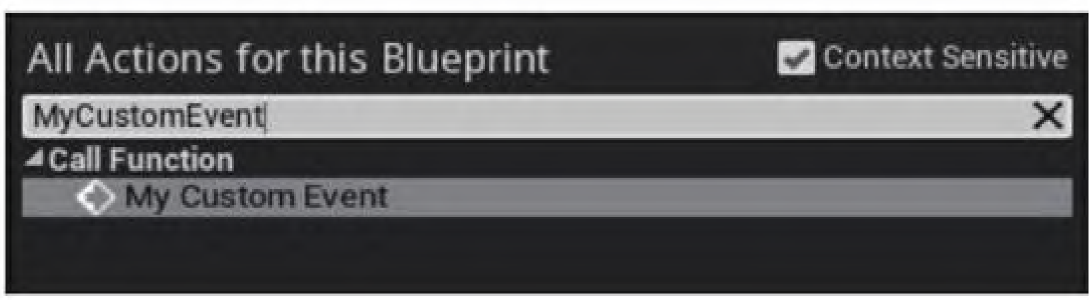



图2-29

图2-30

自定义事件上设置的任何输入参数在新的节点中都将呈现为输入数据引脚，以便它们可以传入自定义事件中。你可以根据需要使用数据连线把任何数据引脚连接到变量或其他数据引脚上。

和常规的Events（事件）不同，常规事件在每个图表中每种事件类型仅能调用一次；但是你可以在图表中多次调用一个自定义事件。这样，自定义事件就可以把多个执行输出分支连接到一个单独的执行输入上，而不需要直接连线，如图2-31所示。

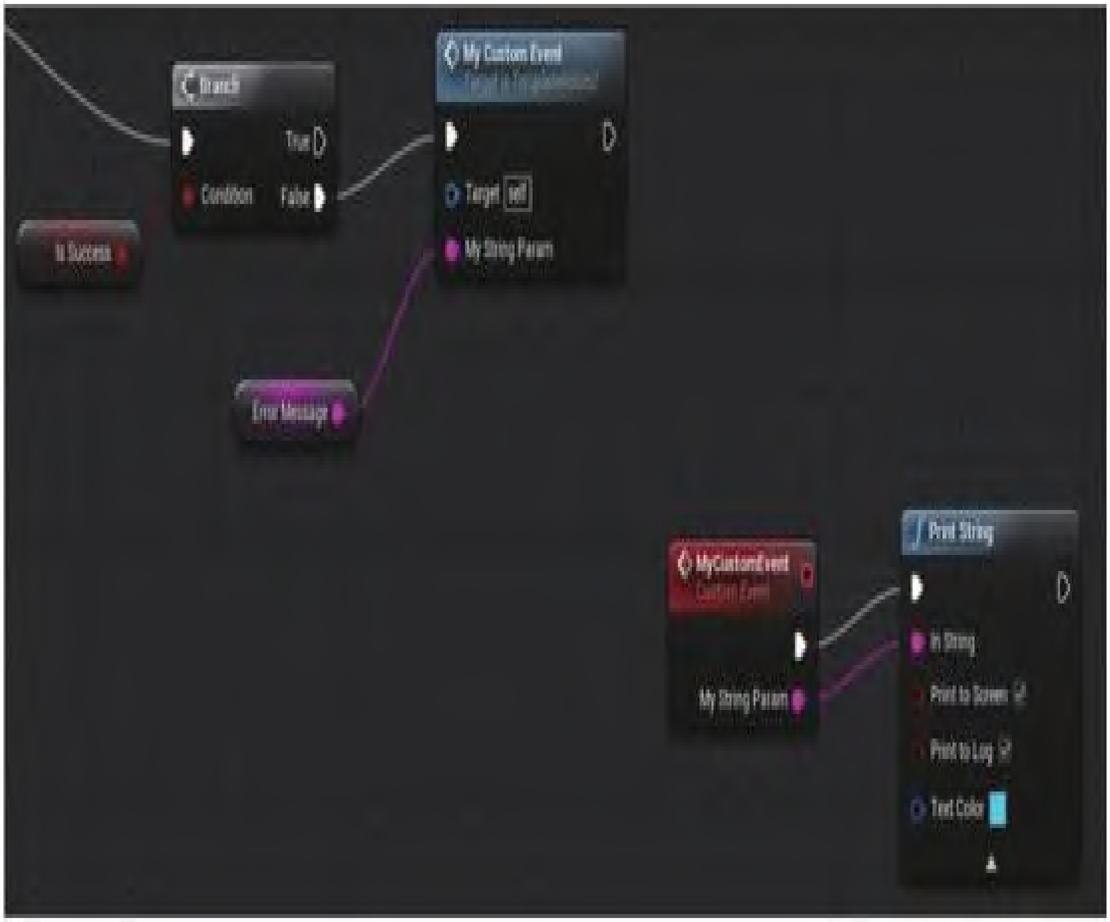



图2-31

在这个示例中，如果IsSuccess布尔变量为false，那么将打印一个错误信息。这个图表的功能，和在序列中的Branch节点后面连接Print String节点的功能一样，但是它有个附加功能是图表中的其他部分可以使用Print String节点，且图表中两个网络部分的彼此位置不必太近。

### 2.3.15.3　自定义事件疑难解答

如果在Custom Event（自定义事件）节点上看到警告“！”条，提示“不能找到名称为[CustomEvent]的函数”信息，请编译你的蓝图，如图2-32所示。

如果你修改了你的自定义事件上的输入参数的数量，那么当你编译蓝图时，调用该自定义事件的所有节点都会出现错误，如图2-33所示。

图2-32

图2-33

你必须刷新调用你的自定义事件的所有节点。要想刷新一个单独节点或者一组选中的节点，请右击这些节点并选择Refresh Nodes（刷新节点），如图2-34所示。

要想刷新你的图表中的所有节点，请在File（文件）菜单中选择Refresh All Nodes（刷新所有节点）项，如图2-35所示。

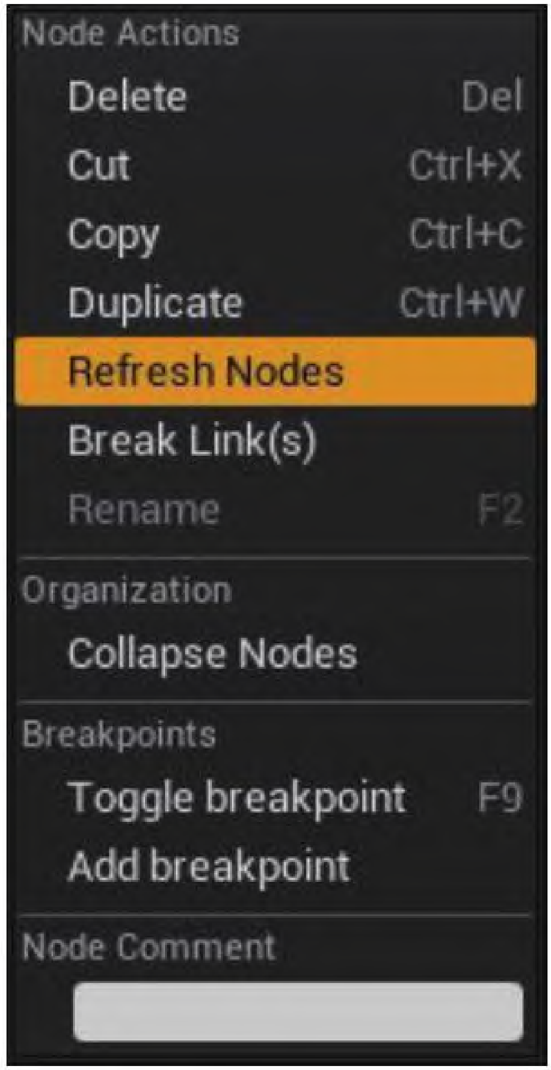

Node Actions
Delete Del
Cut Ctrl+X
Copy Ctrl+C
Duplicate Ctrl+W
Refresh Nodes
Break Link(s)
Rename F2
Organization
Collapse Nodes
Breakpoints
Toggle breakpoint F9
Add breakpoint
Node Comment

图2-34

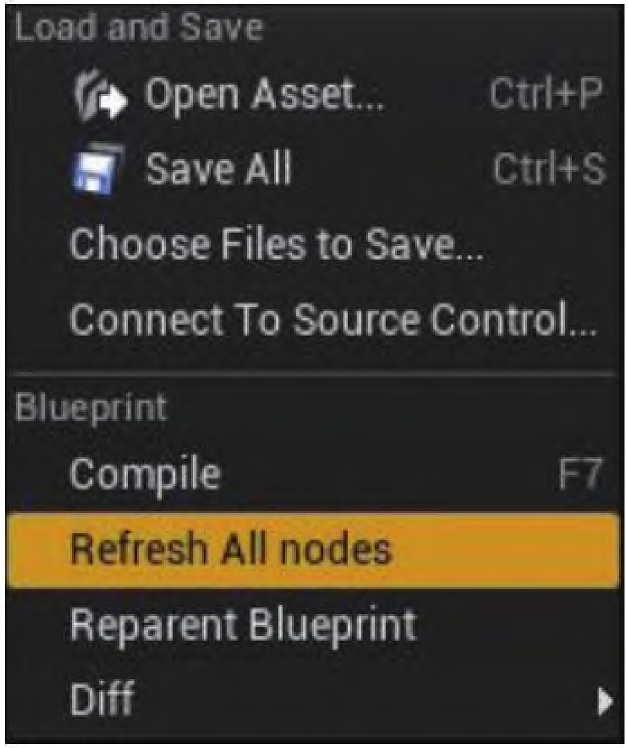

Load and Save
Open Asset... Ctrl+P
Save All Ctrl+S
Choose Files to Save...
Connect To Source Control...
Blueprint
Compile F7
Refresh All nodes
Reparent Blueprint
Diff

图2-35

2.4

# 蓝图中的结构体变量

结构体是相关联的不同数据类型的合集，便于进行访问。你可能已经在蓝图中用到了简单结构体，因为矢量、旋转体和变形均为结构体变量。例如矢量结构体保存彼此关联的X浮点、Y浮点和Z浮点变量。

结构体也可保存其数据。变形结构体保存Actor的位置（矢量结构体）、旋转（旋转体结构体）和大小（矢量结构体）数据。

## 2.4.1 创建结构体

将结构体变量添加到蓝图的方法和添加其他蓝图变量的方法相同。简单结构体（如矢量、旋转体和变形）位列于变量类型下拉菜单的顶部，如图2-36所示。

此下拉菜单还有一个Structure部分，在此可找到蓝图当前可用的全部结构体变量，如图2-37所示。

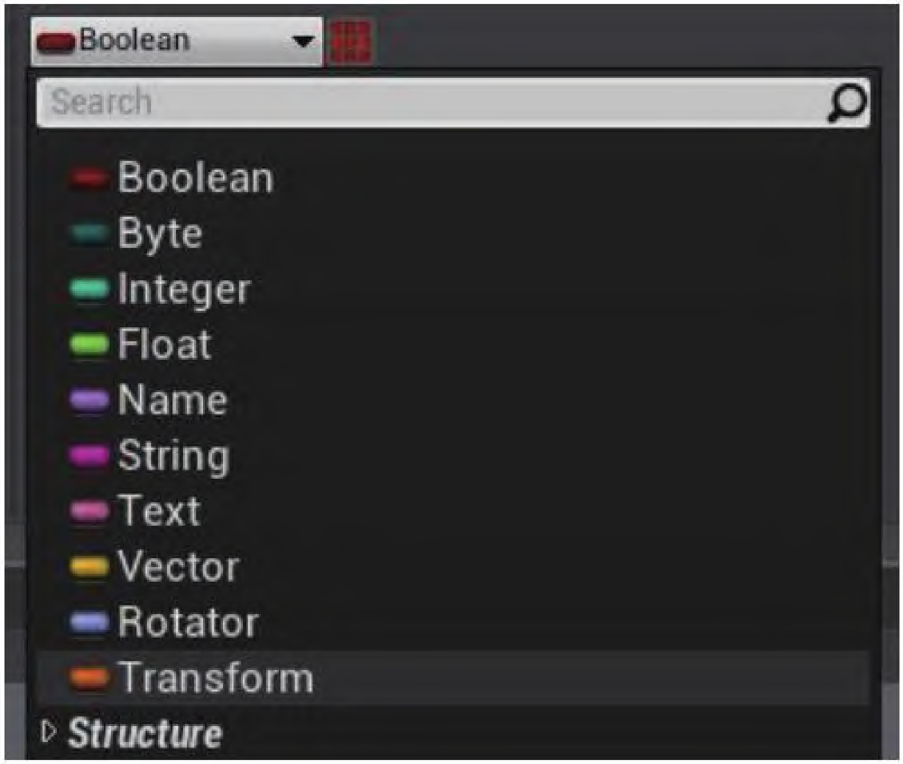

Boolean
Search
Boolean
Byte
Integer
Float
Name
String
Text
Vector
Rotator
Transform
Structure

图2-36

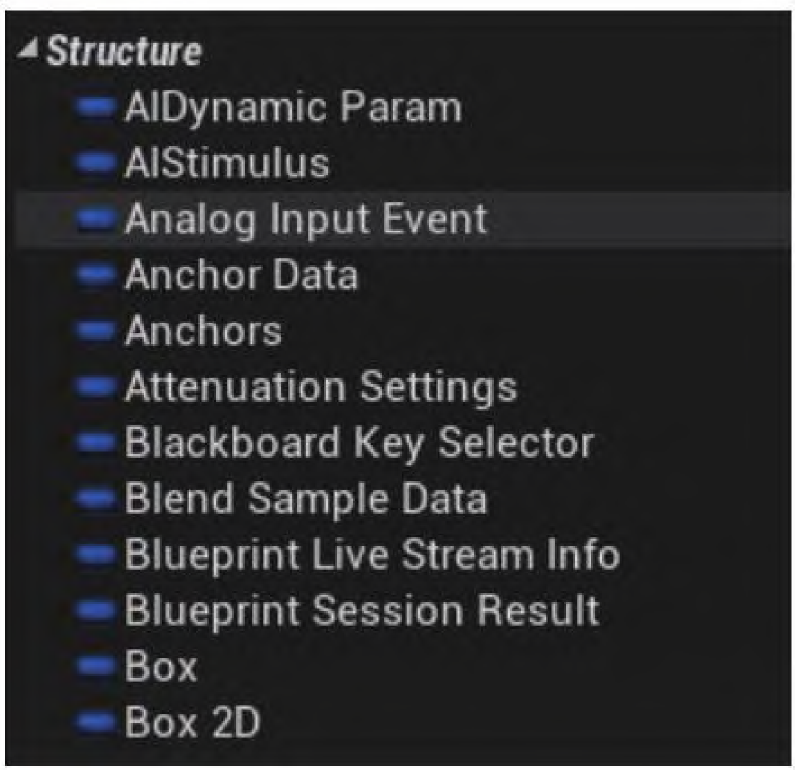

Structure
AIDynamic Param
AIStimulus
Analog Input Event
Anchor Data
Anchors
Attenuation Settings
Blackboard Key Selector
Blend Sample Data
Blueprint Live Stream Info
Blueprint Session Result
Box
Box 2D

图2-37

## 2.4.2　访问结构体信息

结构体的工作是将数据捆绑起来，因此你需要访问那些小块的信息。可通过以下几种不同方法执行。

如需在节点上访问结构体中的单个变量，可使用分离结构体引脚（Split Struct Pin）。

如需分离结构体引脚，可右击引脚并选择Split Struct Pin，如图2-38所示。

这将把结构体中包含的所有变量公开为节点上的单独引脚，便于你输入数值或单独对其进行操作，如图2-39所示。

图2-38

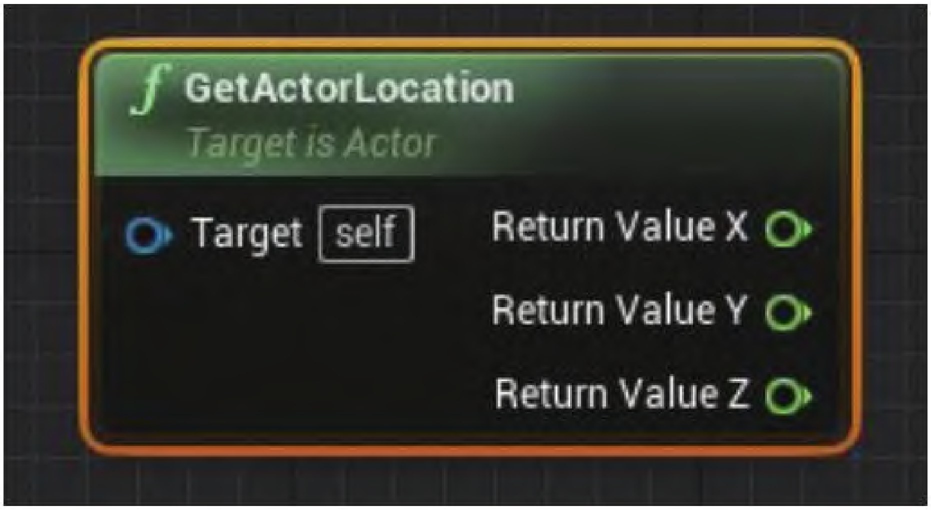



图2-39

如需取消执行Split Struct Pin，可右击任意新引脚并选择Recombine Struct Pin，如图2-40所示。

图2-40

可分离重组输入和输出结构体引脚。

## 2.4.3 拆分结构体

将结构体拆分为单独部分通常是在函数或宏中进行重复的游戏性逻辑。使用Break Struct节点可轻松复制贯穿蓝图图表的行为。如需创建Break Struct节点，可从结构体输出引脚连出引线，从快捷菜单选择Break [Struct Name]，如图2-41所示。

使用的结构体不同，Break Struct节点的命名和输出引脚也有所不同；但总体而言，结构体将被拆分为单独的部分，如图2-42所示。

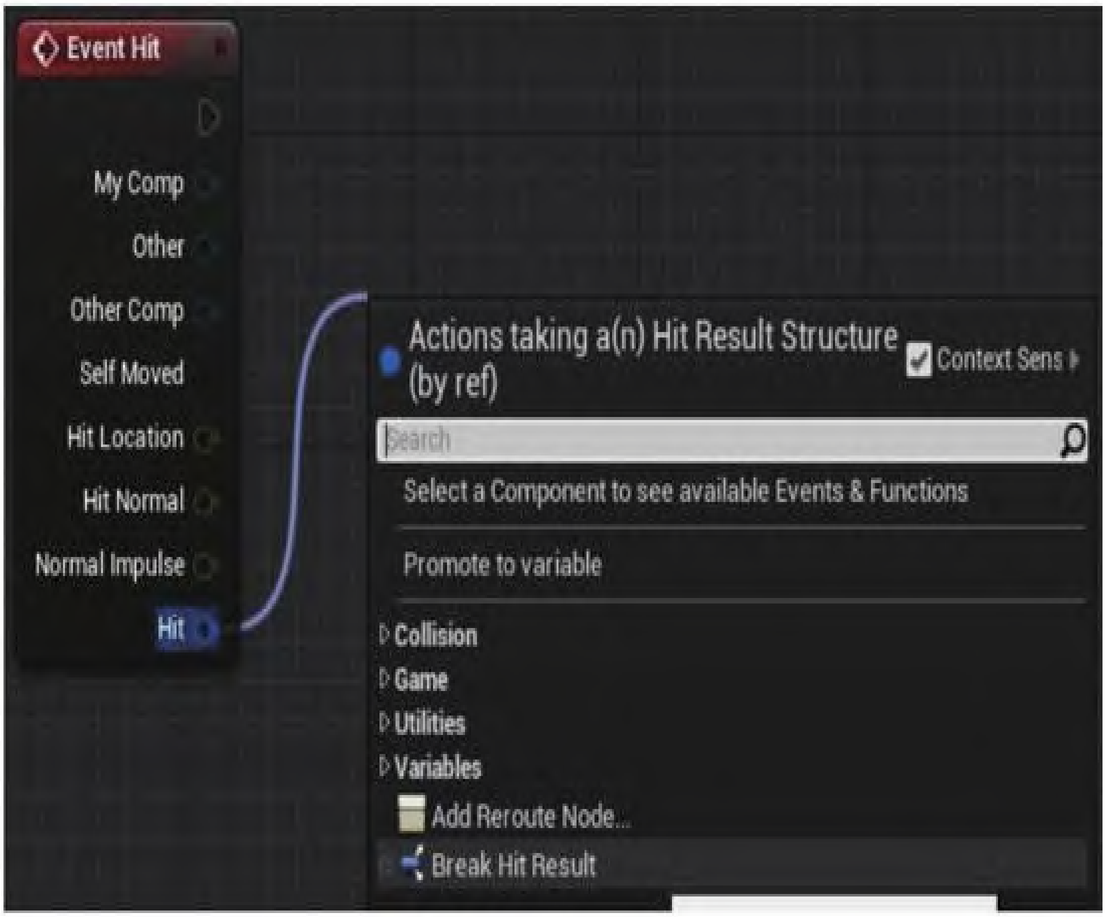



图2-41

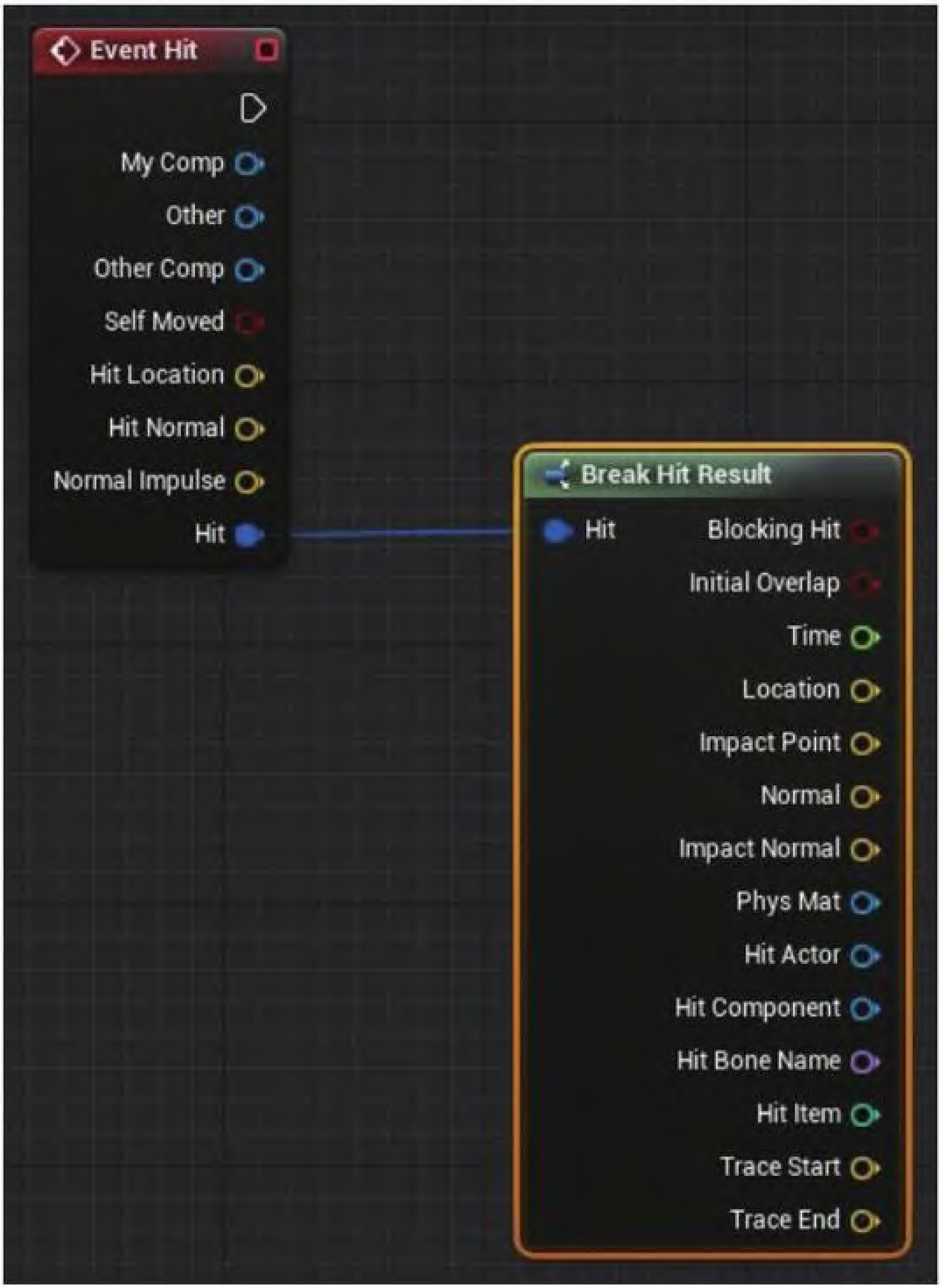



图2-42

举例而言，如需使用Hit Result的Impact Point、Hit Component和Hit Bone Name，可在函数中放置一个Break Hit Result节点，这意味着只需将

Hit Result作为函数输入进行输入，并将这三个数据块在函数中固定保持分离，如图2-43所示。

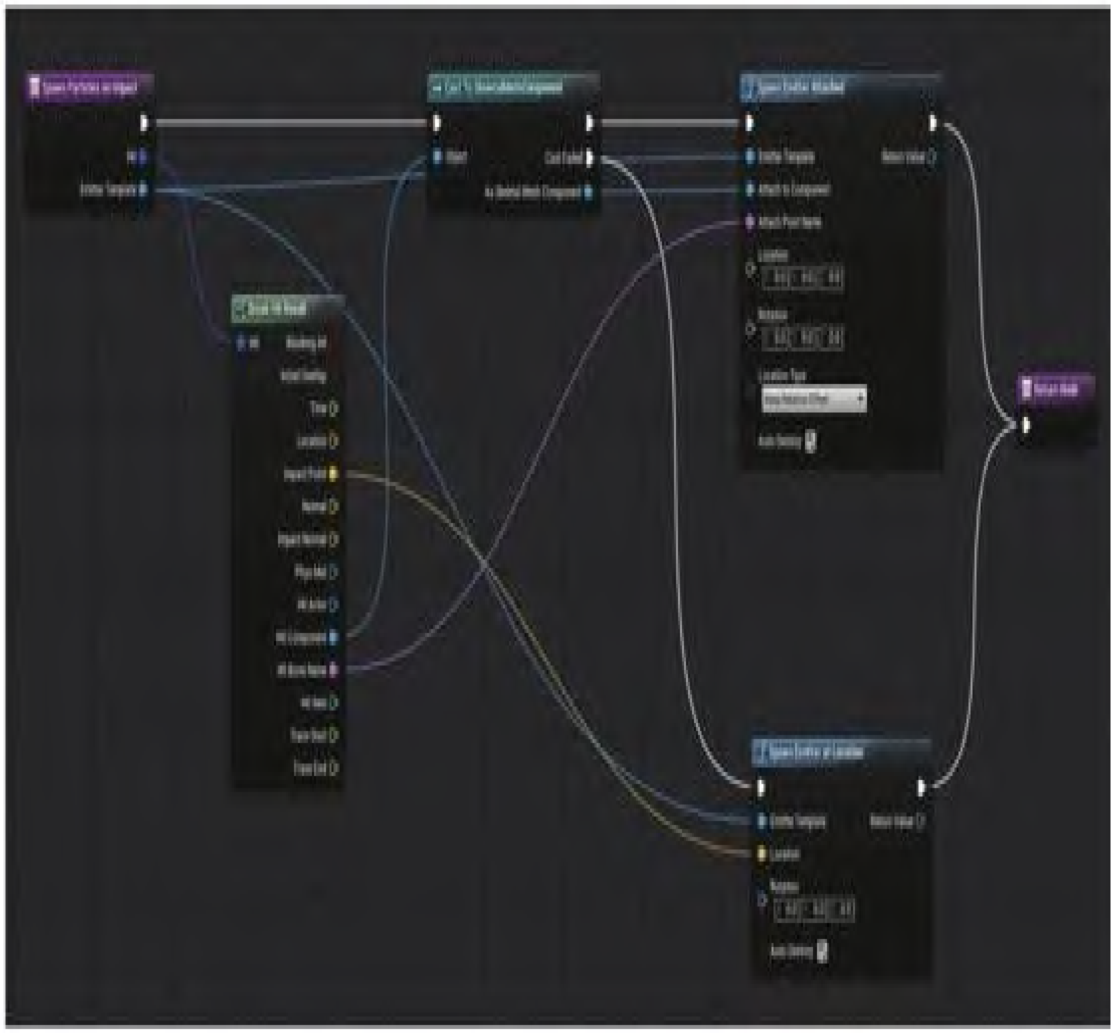


图2-43

### 2.4.3.1　组成结构体

与将结构体筛分为单独数据块相似，也可使用正确的数据组成结构体。

如需创建Make Struct节点，可从结构体输入引脚连出引线，从快捷菜单选择Make[Struct Name]，如图2-44所示。

图2-44

使用的结构体不同，Make Struct节点的命名和输入引脚也有所不同；但总体而言，可通过其包含的所有数据组成结构体，如图2-45所示。

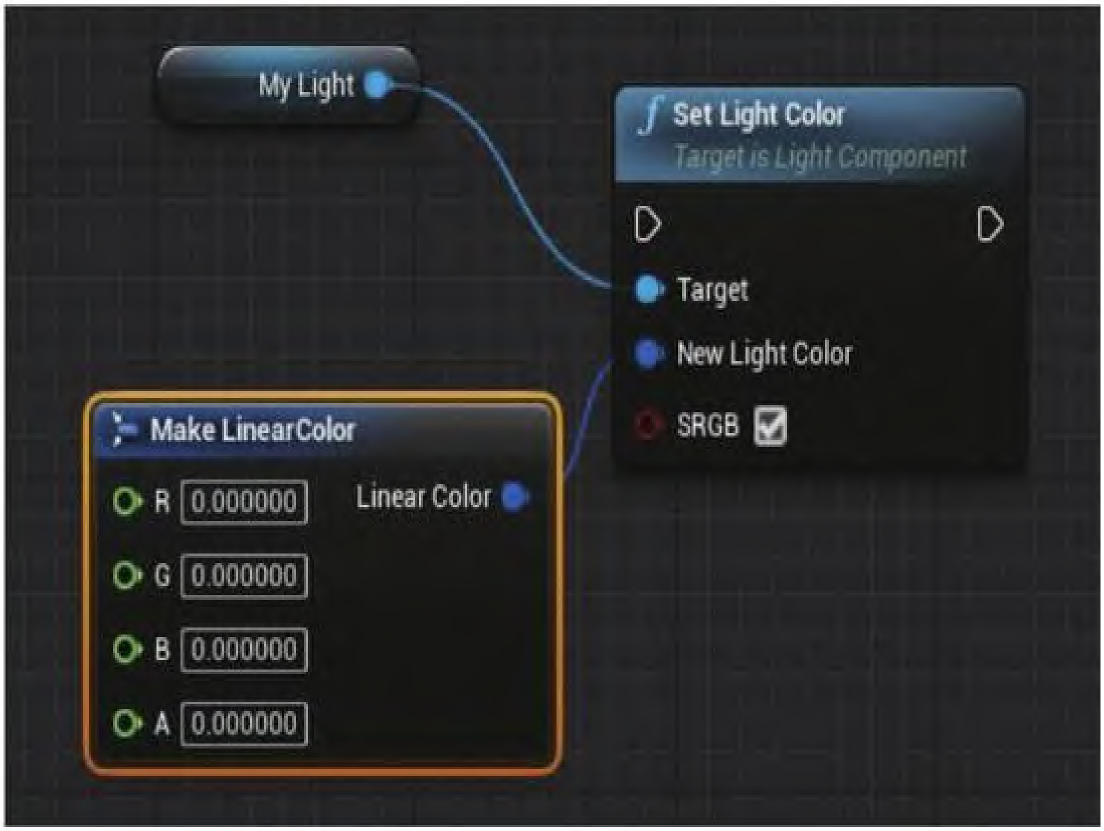



图2-45

### 2.4.3.2　设置结构体中的成员

有时结构体会包含大量数据，而需要修改的只是其中数个元素。对结构体中的成员进行设置即可精确地对数据进行修改，无须将作为固定常量的所有数据引脚连接起来，如图2-46所示。

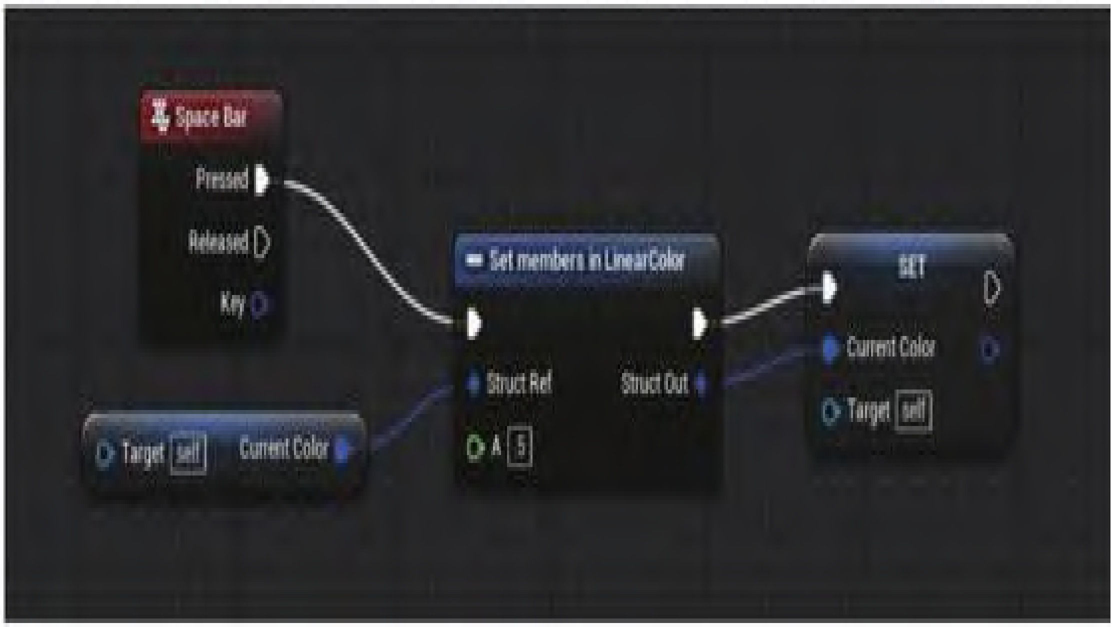
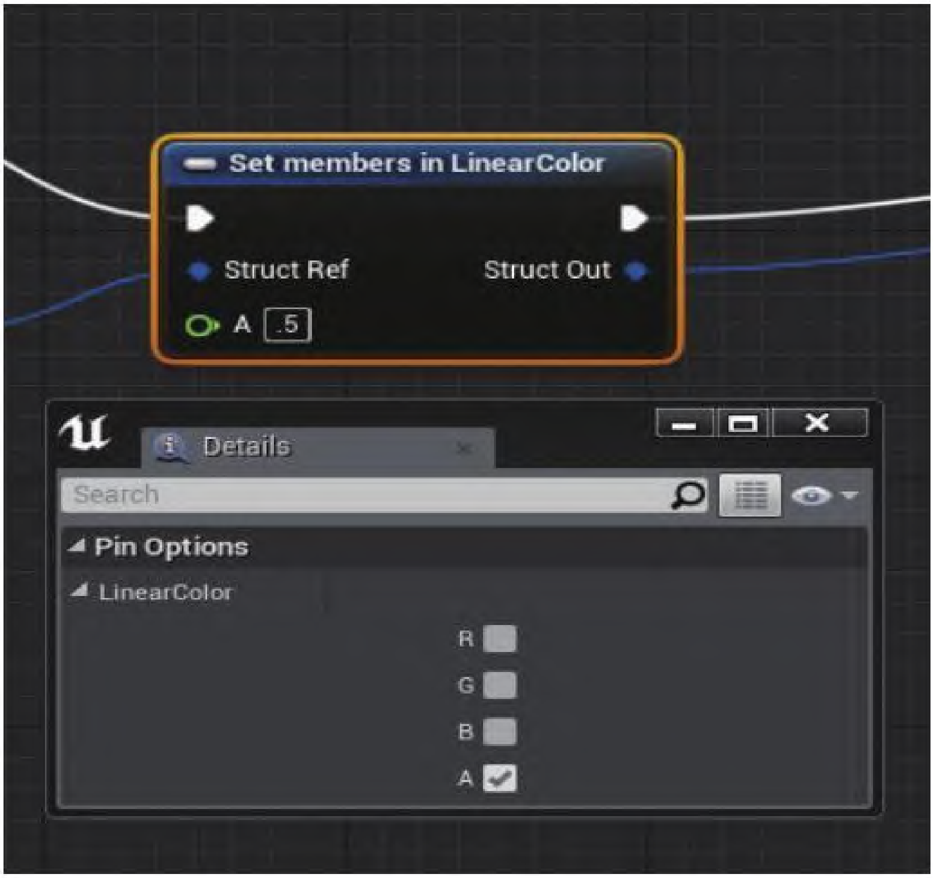



图2-46

如需通过Set Members in Struct节点修改可用成员，要先选择节点。Details中的复选框可将成员作为节点上的引脚公开。未公开的成员变量不会被Set Members in Struct节点修改，如图2-47所示。



Set members in LinearColor
Struct Ref
Struct Out
A
.5
Details
Search
Pin Options
LinearColor
R
G
B
A


图2-47

第3章

# 建造蓝图

3.1

# 蓝图剖析

### 1.蓝图类

Blueprint Class（蓝图类），一般缩写为Blueprint（蓝图），是一种允许内容创建者轻松地基于现有游戏性类添加功能的资源。蓝图是在虚幻编辑器中可视化地创建的，不需要书写代码，会被作为类保存在内容包中。实际上，这些类蓝图定义了一种新类别或类型的Actor，这些Actor可以作为实例放置到地图中，就和其他类型的Actor的行为一样。

### 2.组件

蓝图并不固定包含脚本行为。例如，关卡中的灯柱为不可进行互动的物体，只需要一个网格体代表柱子和灯即可。使用组件搭建可重复使用的蓝图能提高关卡设计的效率。当然，之后可在图表中使用这些组件，使玩家和灯进行互动，或通过时辰系统对它们进行相应的调整。

### 3.图表

图表包含蓝图的设计时行为和游戏时行为。蓝图类实例被创建时构建脚本按组件列表运行，以便你对新对象或Actor的观感进行动态调整。

蓝图中的EventGraph（事件图表）包含一个节点图表，该图表使用事件和函数调用来执行动作，以便对和该蓝图相关的游戏事件做出反应。事件图表通常用于给蓝图的所有实例添加功能。这里也是设置交互性和动态反应的地方。比如，一个光照蓝图可以通过关闭其LightComponent及改变其网格物体使用的材质来对于伤害事件做出反应。这将自动地为该光照蓝图的所有实例提供这种行为。

### 4.组织并重复使用脚本

在蓝图中创建更多脚本后，你会发现脚本的一些段存在经常重复使用的情况。利用函数和宏可对脚本段进行重复使用，但每种方法的优势和使用情况皆不相同。

还可将图表的段合并到嵌套图表中进行整理。

## 3.1.1 蓝图类

蓝图是在虚幻编辑器中可视化地创建的，不需要书写代码，会被作为类保存在内容包中。实际上，这些类蓝图定义了一种新类别或类型的Actor，这些Actor可以作为实例放置到地图中，就和其他类型的Actor的行为一样。

### 3.1.1.1 Parent Classes

你可以创建多种不同类型的蓝图，当然在你做这些之前，你需要指定该蓝图的Parent Class（父类）选择继承父类，允许你在自己的蓝图里面调用父类创建的属性。

下面是你创建蓝图是最常见的父类：

| 类　型 | 描　述 |
|---|---|
| Actor | 一个可以在世界中摆放，或者生成的 Actor |
| Pawn | Pawn 是一个可以从控制器获得输入信息处理的 Actor |

续表

| 类　型 | 描　述 |
|---|---|
| Character | 角色是一个包含了行走，跑步，跳跃以及更多动作的 Pawn |
| PlayerController | A Player Controller is an Actor responsible for controlling a Pawn used by the player |
| Game Mode | 一个 Game Mode 定义了游戏是如何被执行的，游戏规则，如何的分以及其他方面的内容 |

上面的是最常用的内容，所有的类都可以为一个新建的蓝图用作父类（甚至其他的蓝图类）。

例如，你创建了一个叫做Animals的Actor Blueprint在它里面实现了所有动物的共享属性，例如__Hunger，Thirst，Energy，或者是其他你想实现的脚本。然后你从_Animals_作为父类，创建了另外一个叫做_Dog_的蓝图；你在这个Dogs_里定义了例如_Play Fetch or Roll Over之类特别的功能，那么这个_Dogs_除了独享自己的功能之外，还继承了_Animals里定义的其他功能。

根据所用的方法create a Blueprint Class，一个父类将在被创建时分配。

### 3.1.1.2 用蓝图类工作

参考下面的信息蓝图类，如图3-1所示。

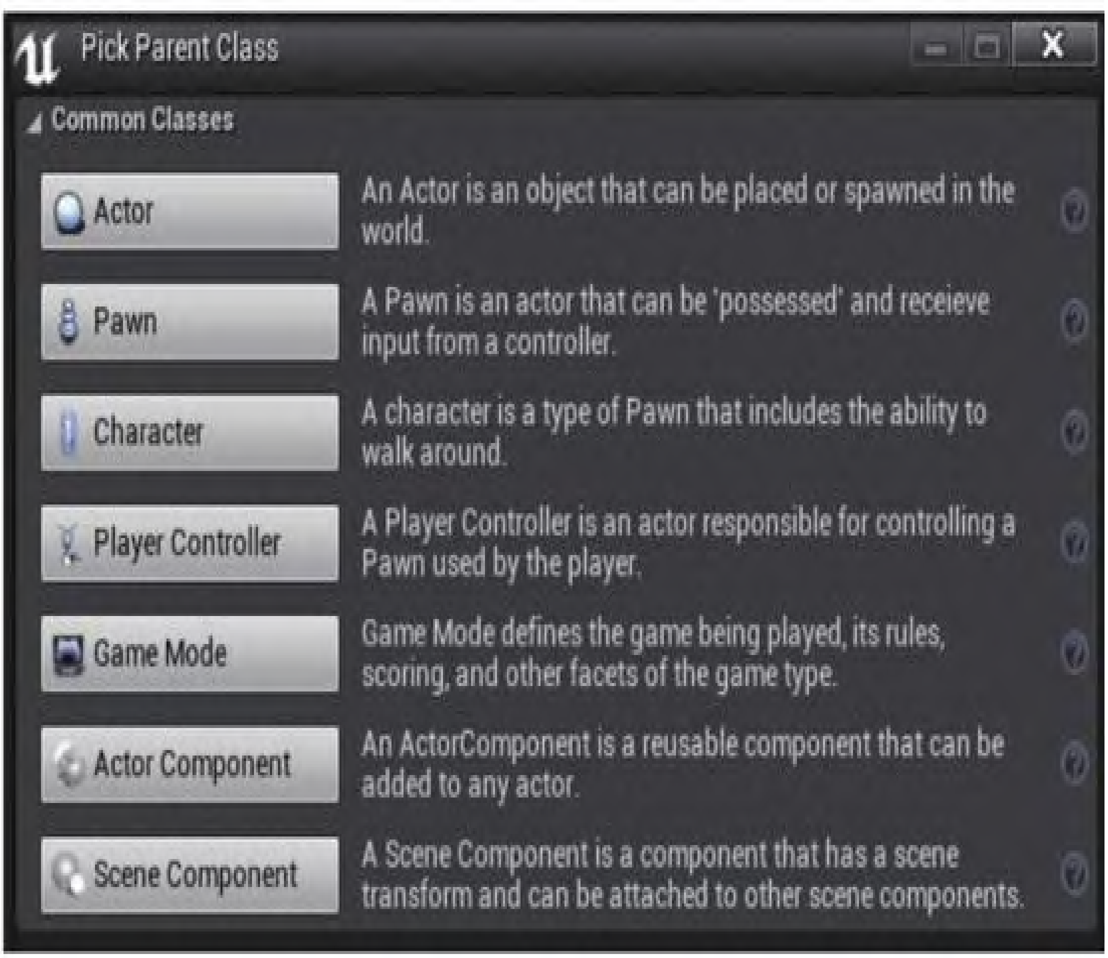



图3-1

打开蓝图类，如图3-2所示。

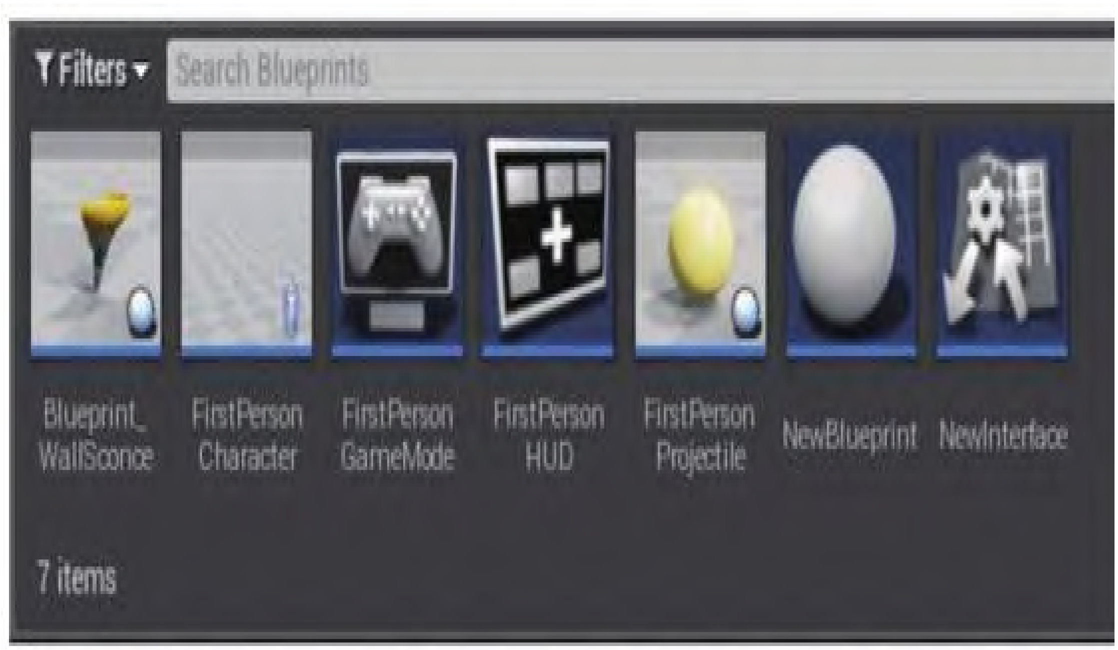



图3-2

编辑蓝图类，如图3-3所示。

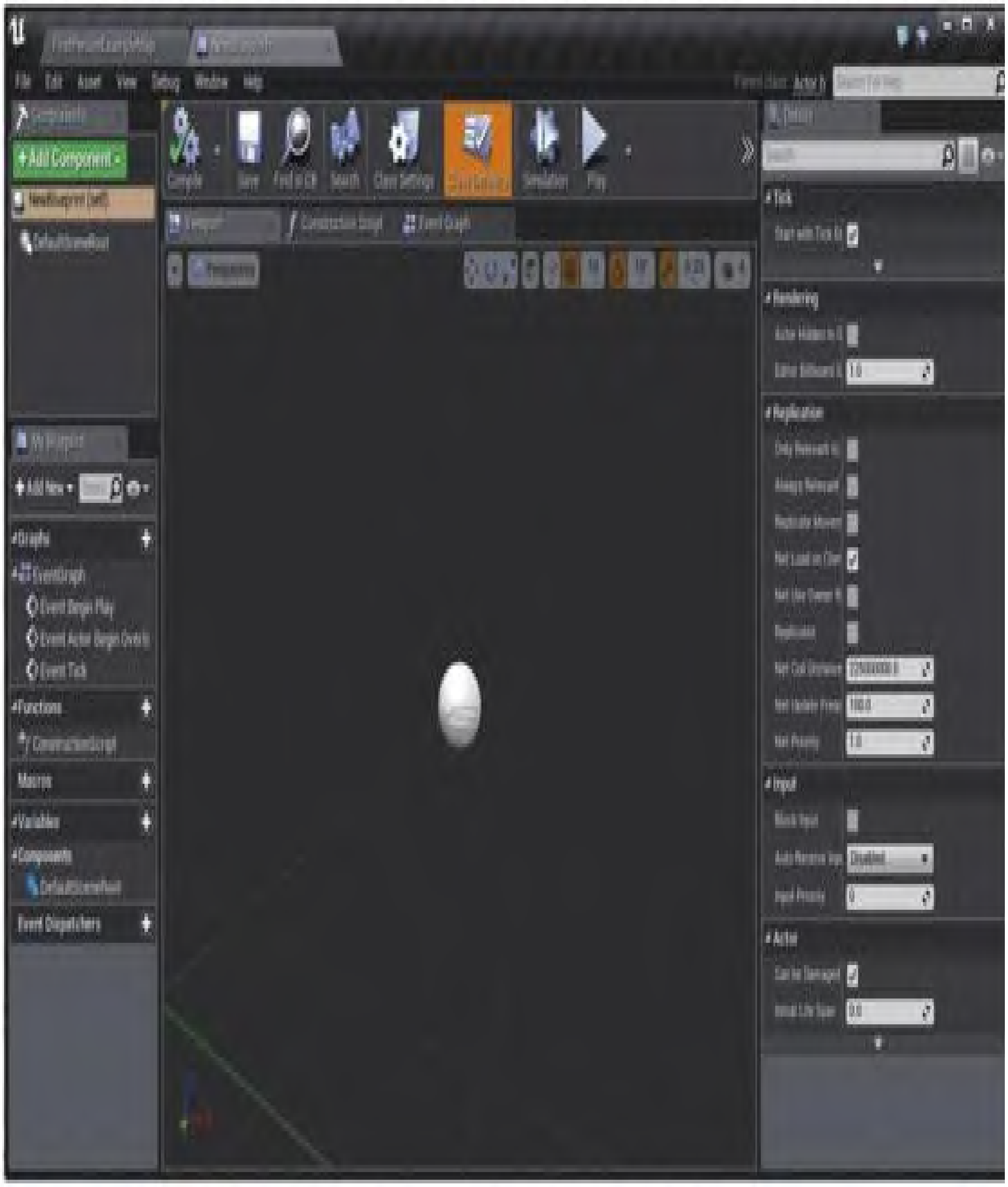



图3-3

蓝图编辑器总览，如图3-4所示。

图3-4

### 3.1.1.3　仅包含数据的蓝图

Data-Only Blueprint（仅包含数据的蓝图）是指仅包含代码（以节点图表的形式）、变量及从其父类继承的组件的类蓝图。仅包含数据的蓝图允许你调整及修改继承的属性，但是不能添加新元素。从本质上讲，这些蓝图是原型的替代物，设计人员可以使用它们来调整属性或者设置具有变种的项目。

Data-Only Blueprints（仅包含数据的蓝图）是在合并的属性编辑器中进行编辑的，但是也可以通过使用完整的蓝图编辑器来添加代码、变量或组件，将其转换为完整的蓝图。

## 3.1.2 蓝图基础用户指南

本节含蓝图基础使用案例和常用操作，助你快速上手。

### 3.1.2.1 创建蓝图

有数种创建蓝图的方法，第一种是使用Content Browser中的Add New按钮，如图3-5所示。

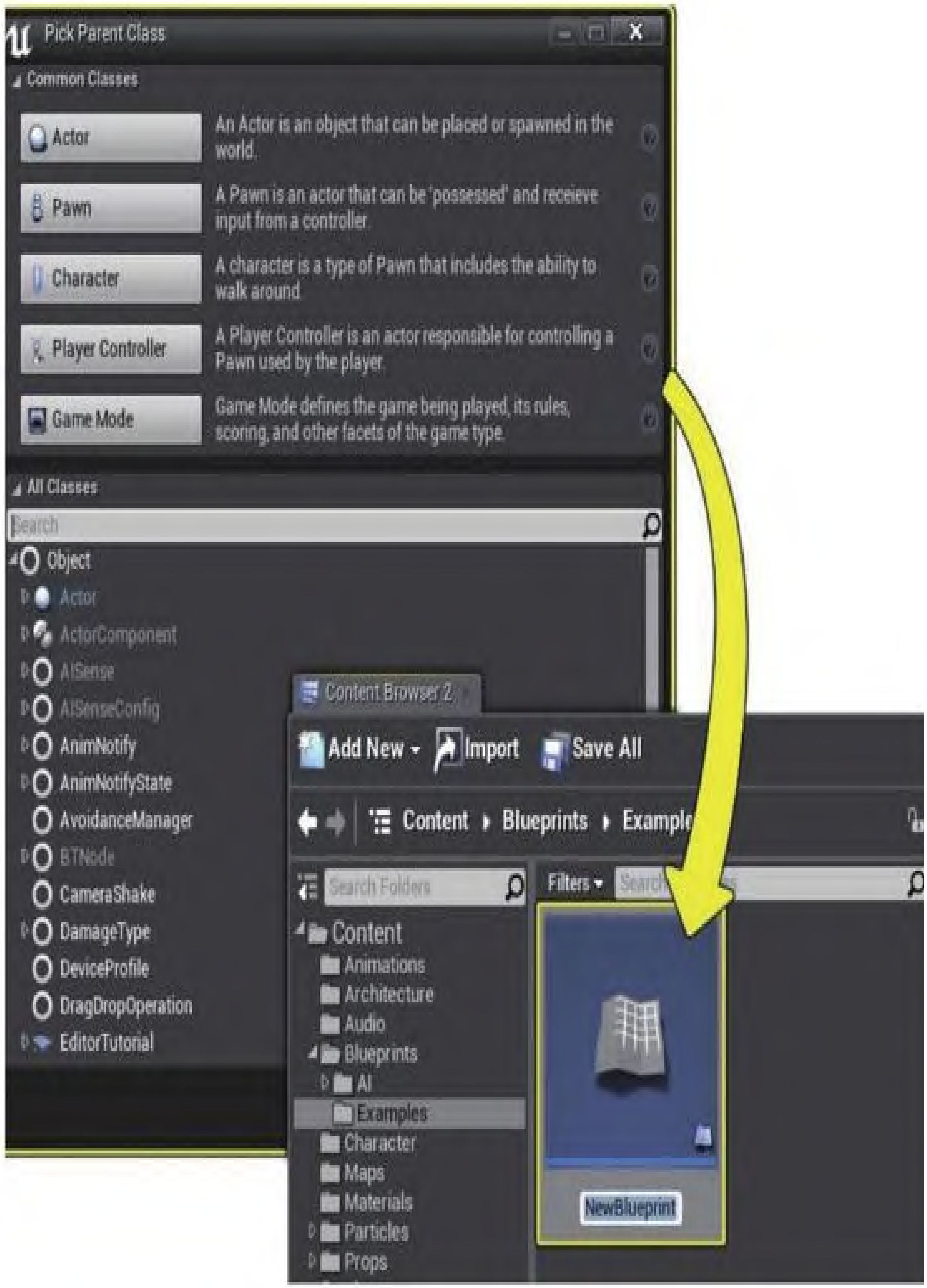
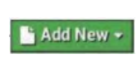



图3-5

Step 1　在Content Browse里面，单击 

 。

Step 2　在出现的菜单下面选择Blueprint。

额外的蓝图类型可以在Create Advanced Asset内的Blueprints选项中创建。

Step 3　为你的蓝图选择一个Parent Class，如图3-6所示。

选择类后，一个新的蓝图资源将被添加至Content Browser，然后可为其指定一个名称，如图3-7所示。

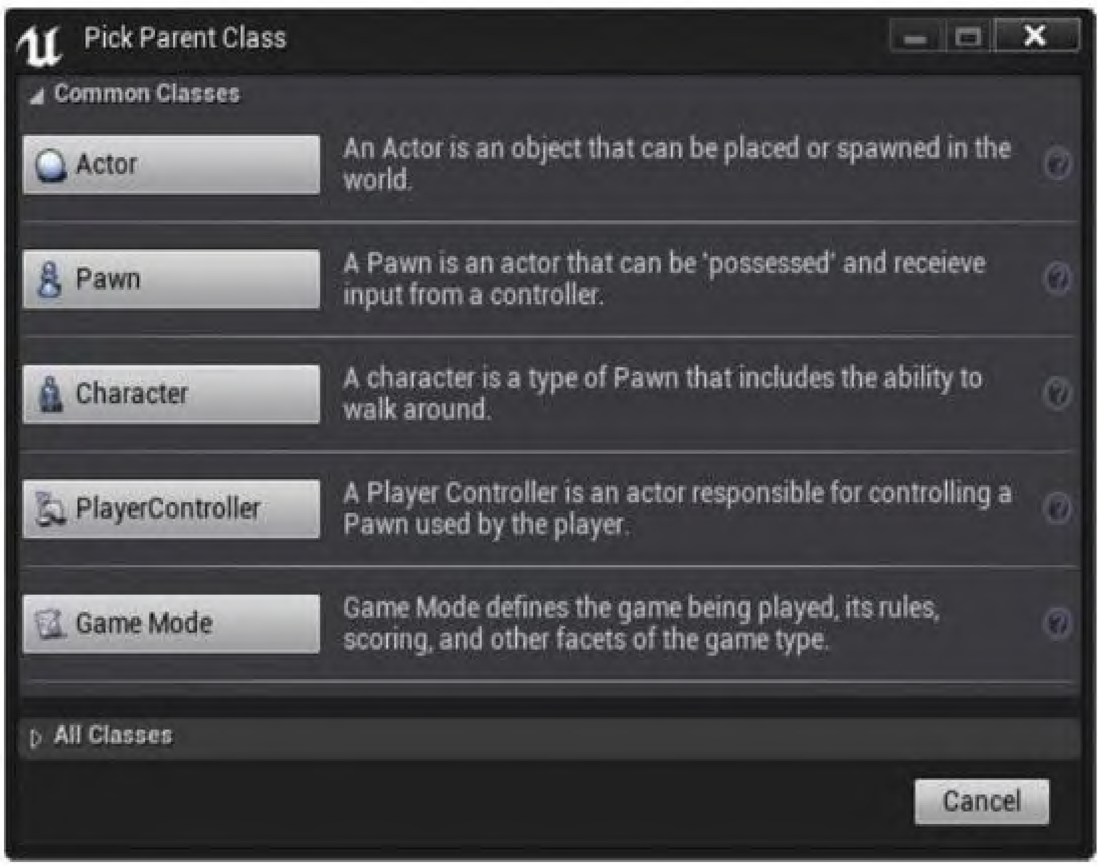



图3-6

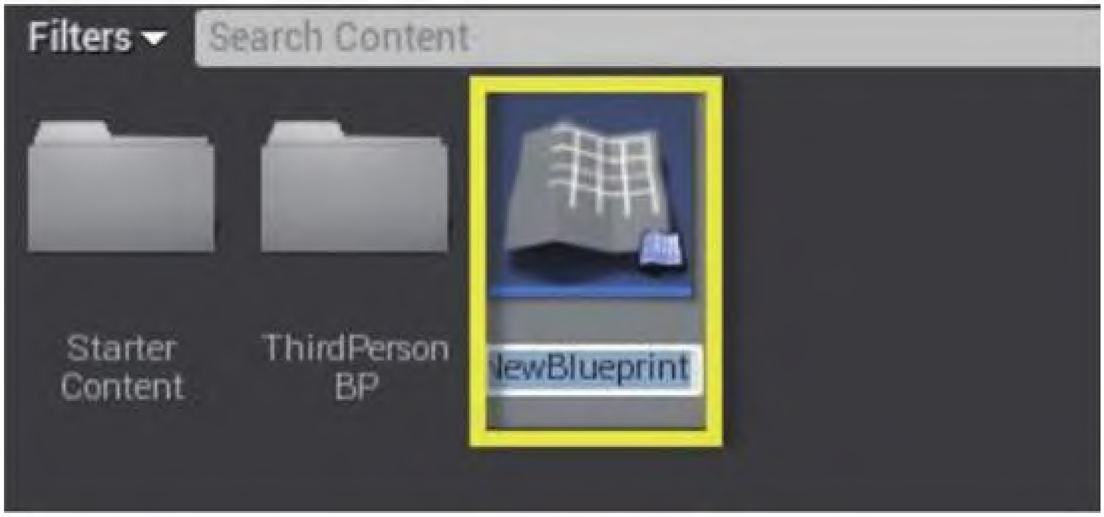



图3-7

创建蓝图使用资源：

可在Content Browser中的资源上右击创建一个蓝图，然后在Asset Actions下选择Create Blueprint Using This选项，如图3-8所示。

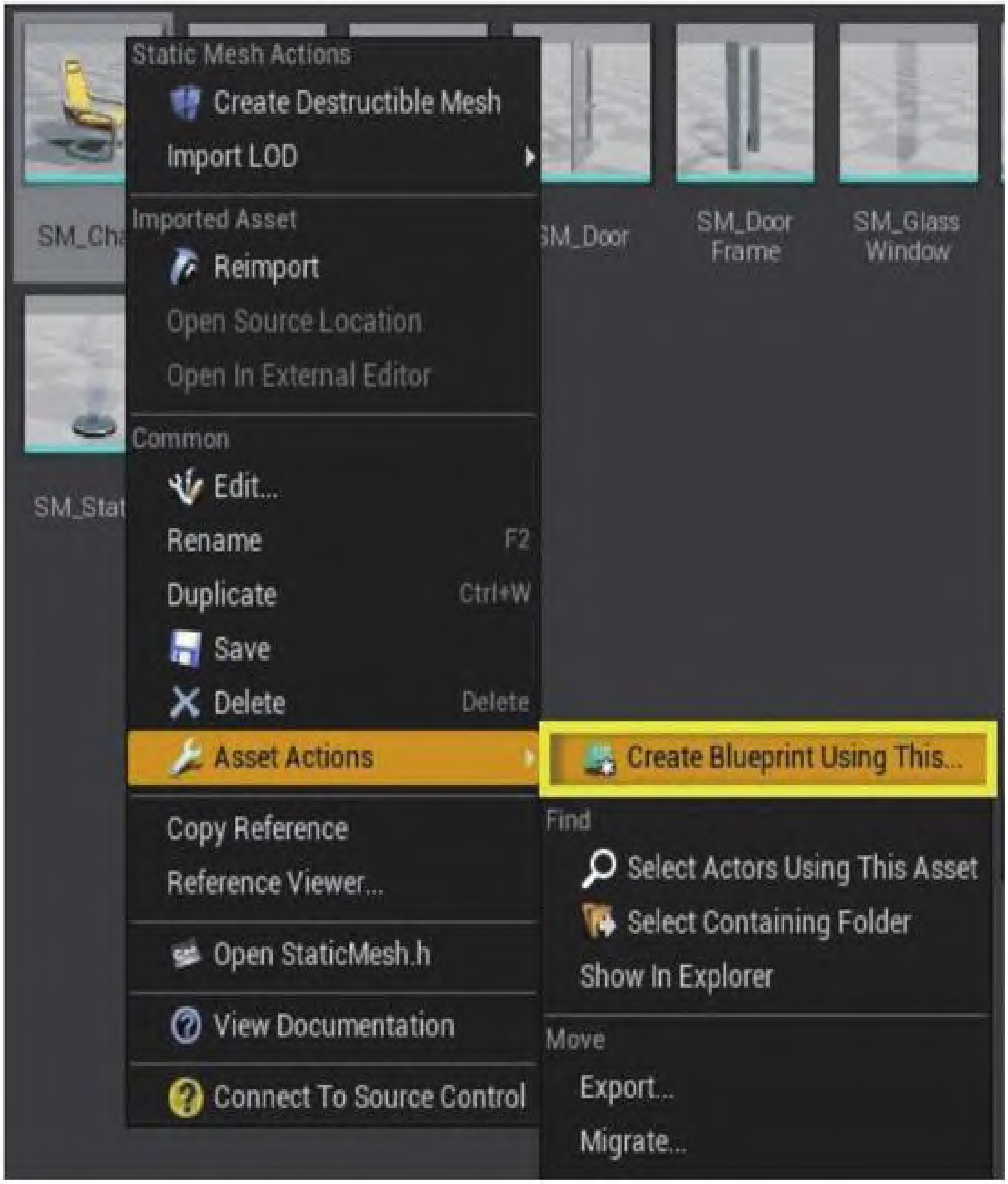



图3-8

该选项仅可用支持的资源为静止网格物体、骨骼网格物体、粒子效果、声音提示或声波等。如该项为灰色，则表示它在所选资源上不可用。

选择Create Blueprint Using This选项后，将弹出提示让你选择蓝图的保存位置。确认保存位置后，蓝图将在蓝图编辑器中自动打开。

### 3.1.2.2 在关卡中放置蓝图

在关卡中放置蓝图有以下方法：

从Content Browser将其拖放进关卡，如图3-9所示。

或在Content Brower中选择蓝图后，在关卡中右击并从快捷菜单中选择Place Actor，如图3-10所示。

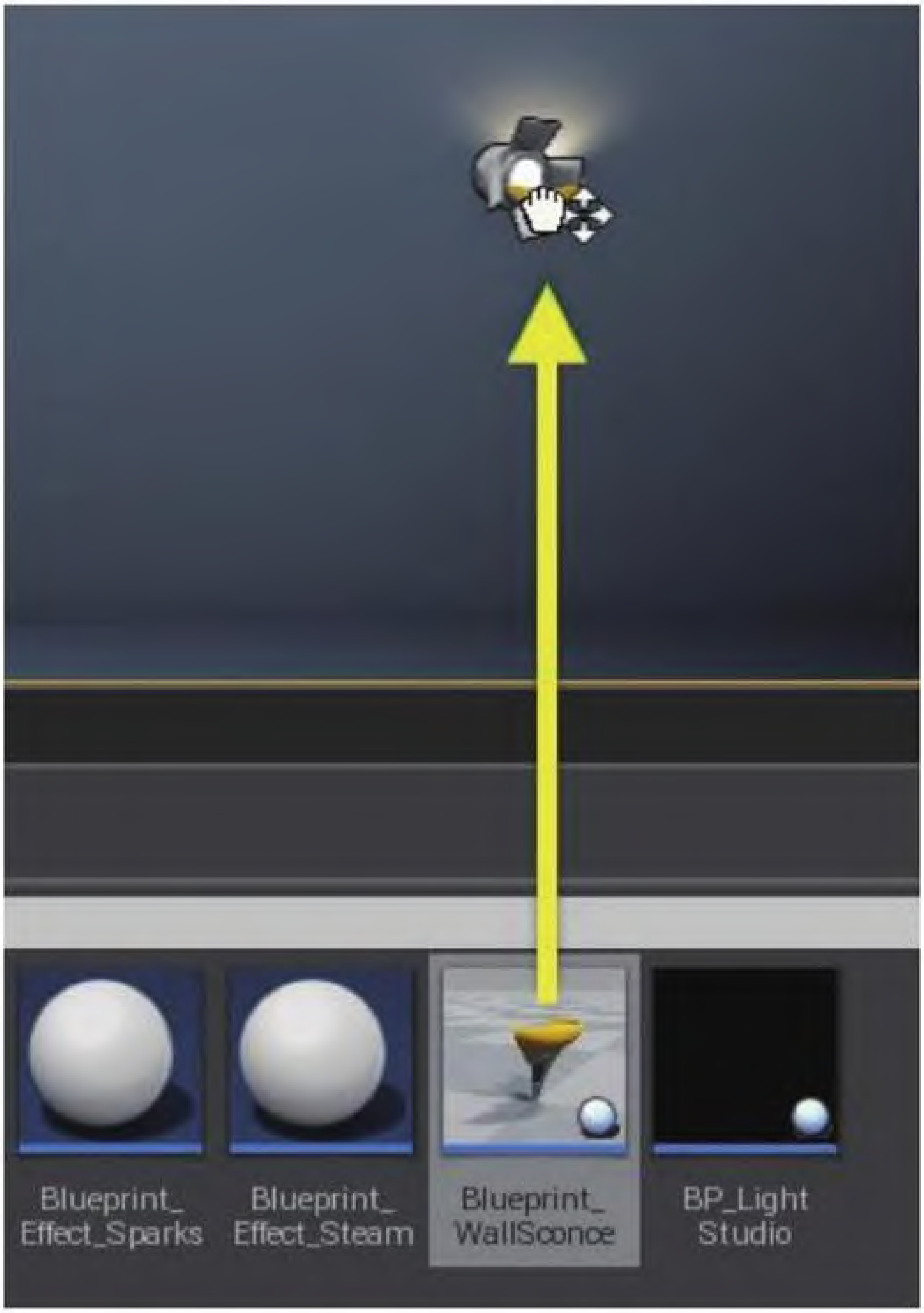

Blueprint_
Effect_Sparks
Blueprint_
Effect_Steam
Blueprint_
WallSconce
BP_Light
Studio

图3-9

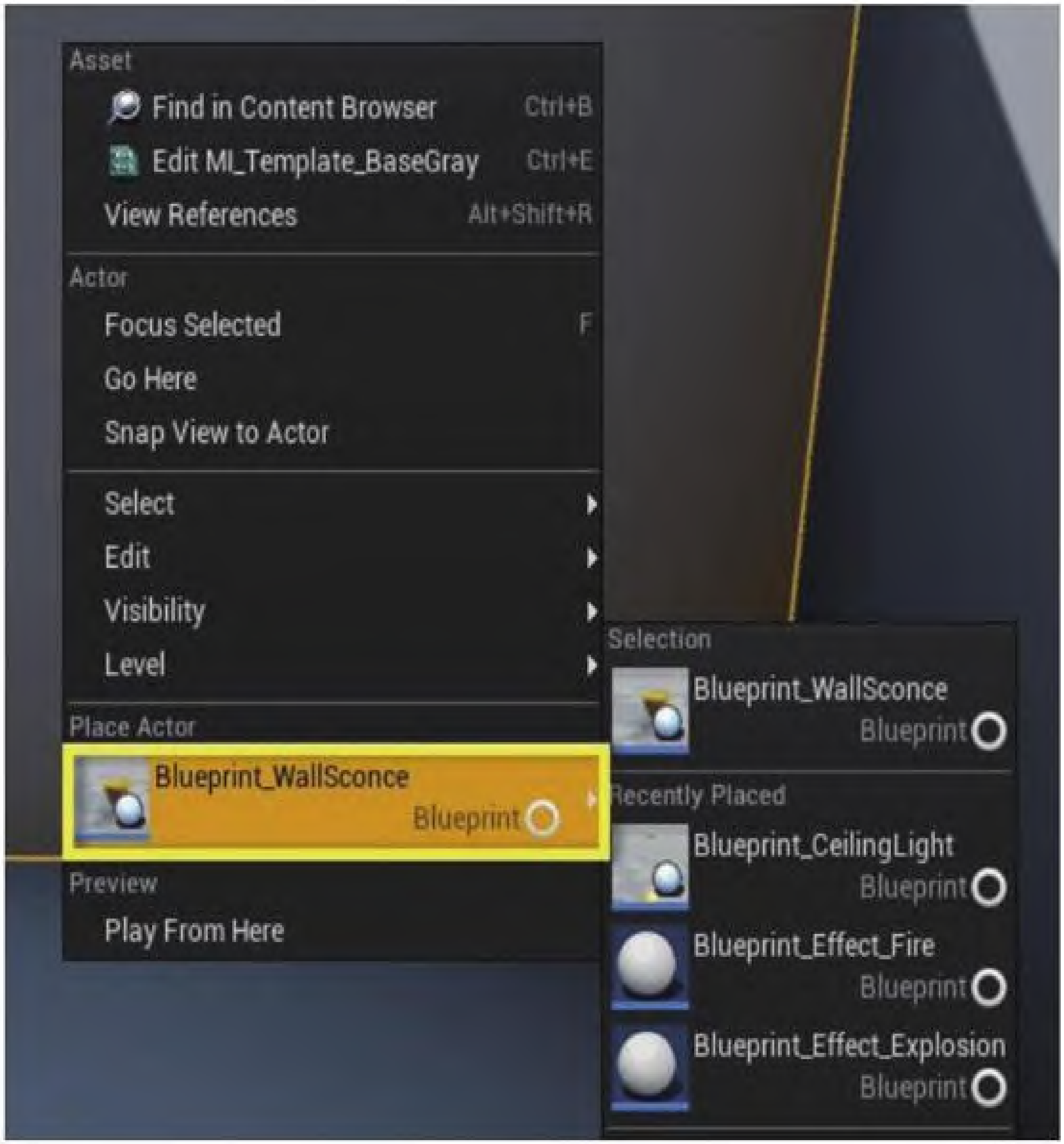



图3-10

### 3.1.2.3　放置蓝图节点

在图表模式中放置节点的方法有多种（详情请查阅放置节点），本章节将讲述最常用的方法，以及如何连接节点。

多数情况下，可在蓝图图表中右击访问快捷菜单放置节点，如图3-11所示。

图3-11

从上图菜单中展开任意类目（或子类目），然后选择需要的节点添加至图表中。

窗口右上角有一个名为Context Sensitive的选项。它为默认开启，禁用此选项后将基于当前上下文自动筛选菜单中显示的选项，如图3-12所示。

图3-12

Context Sensitive选项开启时右击并搜索Animation，便会出现筛选列表，如图3-13所示。

然而，如取消勾选Context Sensitive并搜索Animation，便会出现所有与animation相关的内容，如图3-14所示。

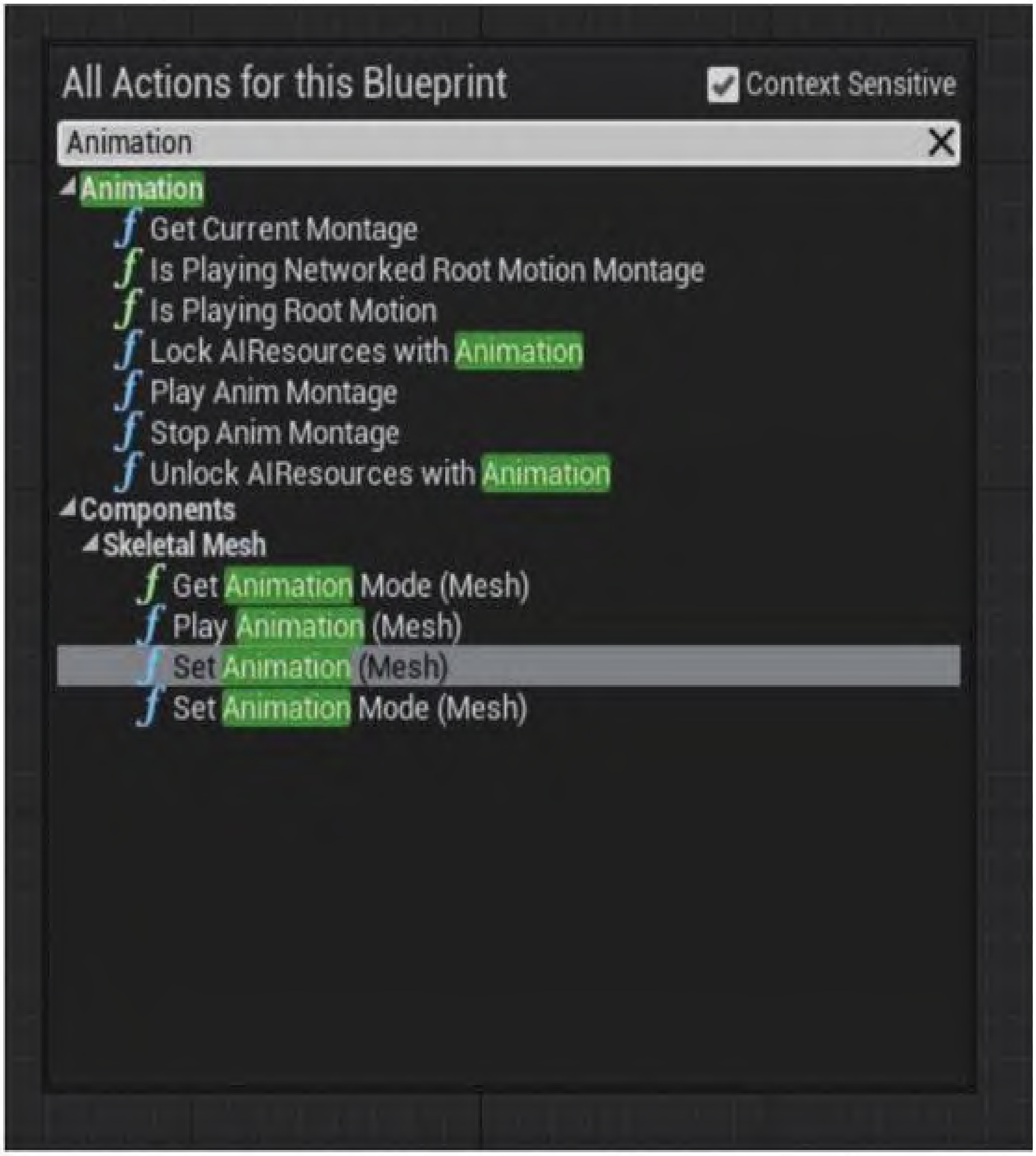

All Actions for this Blueprint
Context Sensitive
Animation
Animation
Get Current Montage
Is Playing Networked Root Motion Montage
Is Playing Root Motion
Lock AIResources with Animation
Play Anim Montage
Stop Anim Montage
Unlock AIResources with Animation
Components
Skeletal Mesh
Get Animation Mode (Mesh)
Play Animation (Mesh)
Set Animation (Mesh)
Set Animation Mode (Mesh)

图3-13

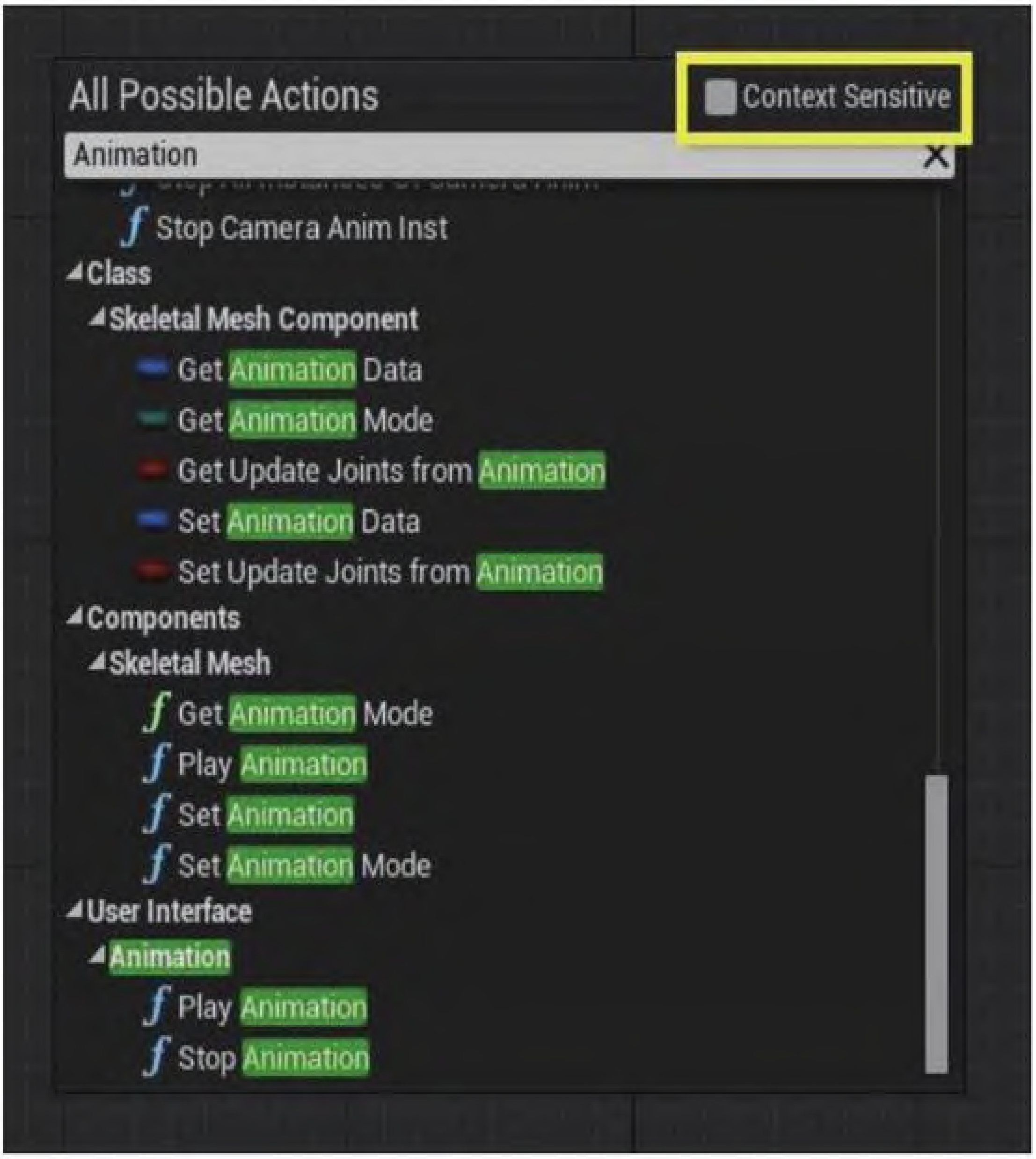



图3-14

图表中右击呼出快捷菜单，也可拖动现有节点访问快捷菜单，如图3-15所示。

图3-15

在上图中有一个Character Movement组件引用，拖动其输出引脚可添加连接上下文的节点。这些节点和被拖动的节点为相关，如图3-16所示。

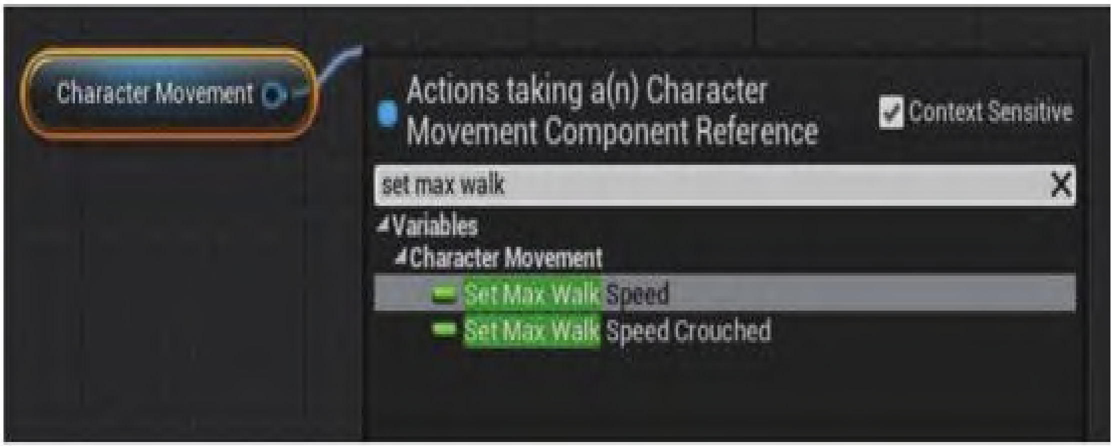



图3-16

在上图中，搜索Set Max Walk，然后从菜单中选择Set Max Walk Speed对角色的最高步行速度进行设置。

#### 3.1.2.4　连接蓝图节点

如需连接节点，拖动一个引脚并将其连接至另一个同类引脚（在某些情况下会创建一个转换节点。举例而言，将一个Float输出引脚与一个Text输入引脚相连，这两者间将创建一个转换节点并自动转换和连接两个节点）。

下图显示的是两个节点的基本连接，输入/输出引脚类型相同，如图3-17所示。

以下实例则是转换进行中的状态，如图3-18所示。

图3-17

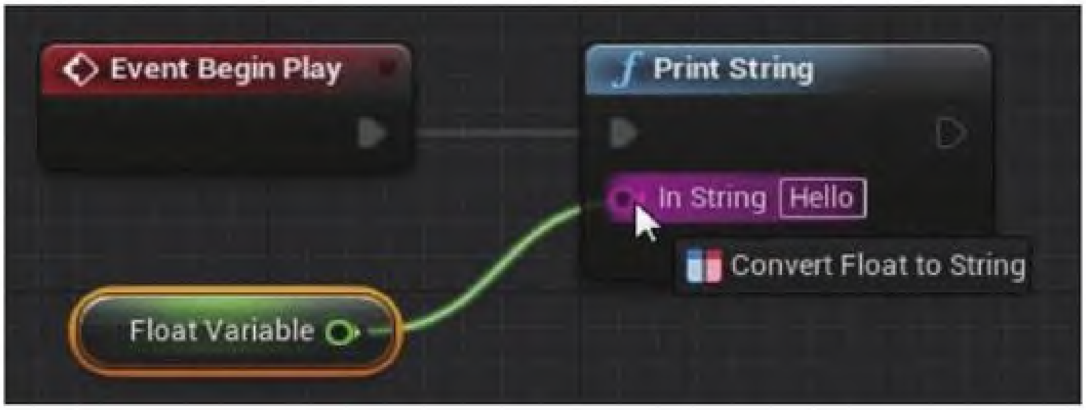
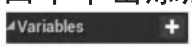



图3-18

### 3.1.2.5 创建变量

Variables（变量）是存放一个值或引用世界中的一个Object或Actor的属性。这些用户界面内部访问，或者通过设置使得可以在外部进行访问，以便应用放置在关卡中的蓝图实例的设计人员可以修改它们的值。

可在MyBlueprint窗口中单击添加按钮为蓝图创建变量。 on the variable list header 

。

创建变量后，需要定义变量的属性。

### 3.1.2.6 创建变量

在我的蓝图中，通过使用

按钮可以直接添加新变量。你可以在我的蓝图选卡中命名该变量，并在详细信息选卡中设置其类型及其他属性，如图3-19所示。

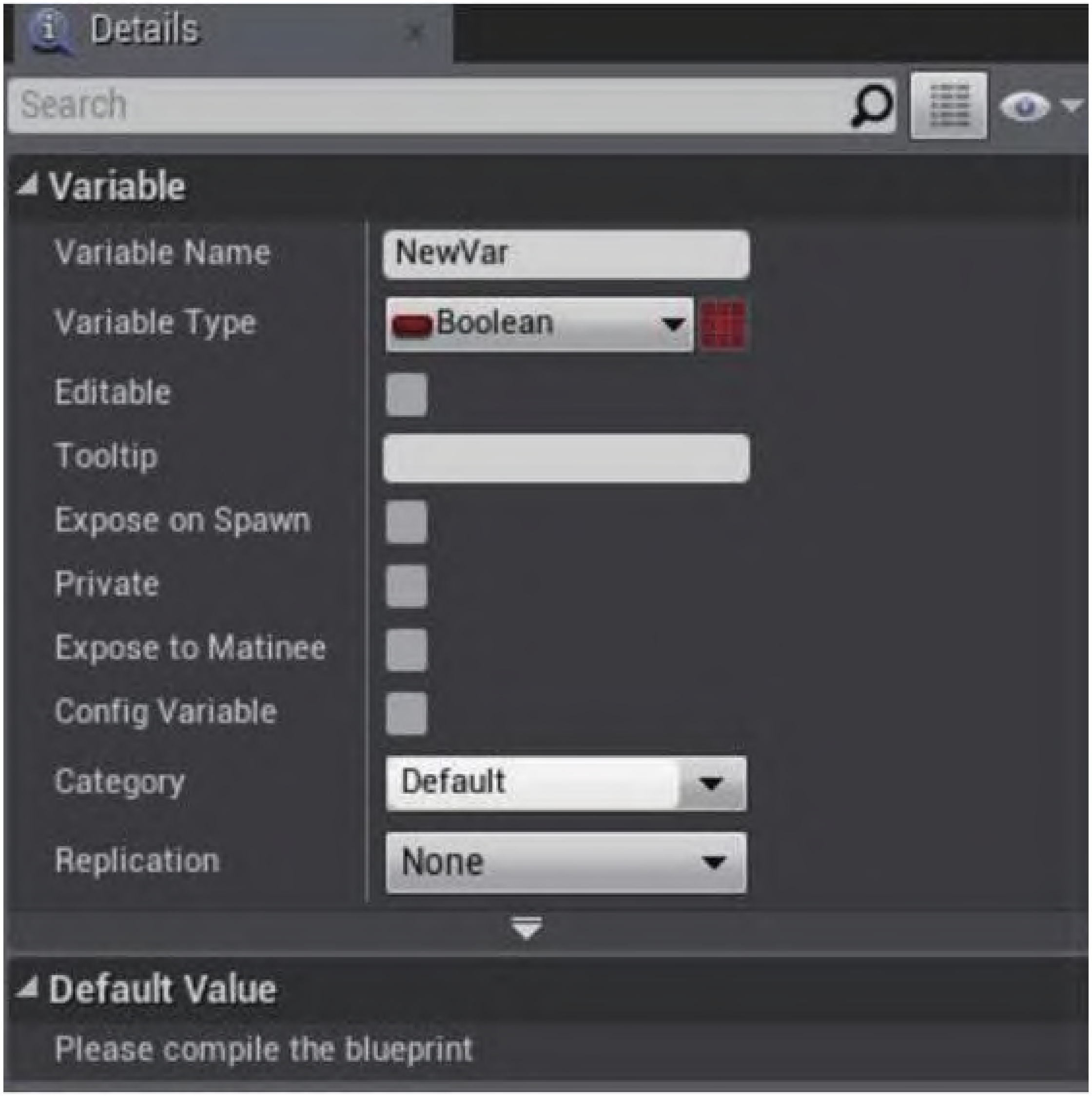



图3-19

### 1. 添加一个新变量

**Step 1**　在我的蓝图选卡内，在文本框中输入该变量的名称，并按下回车键，如图3-20所示。

**Step 2**　在详细信息选卡中，从Variable Type（变量类型）菜单中选择你想添加的变量的类型，如图3-21所示。

**Step 3**　某些变量类型需要展开来查看需要的子菜单项，比如对象或结构体，如图3-22所示。

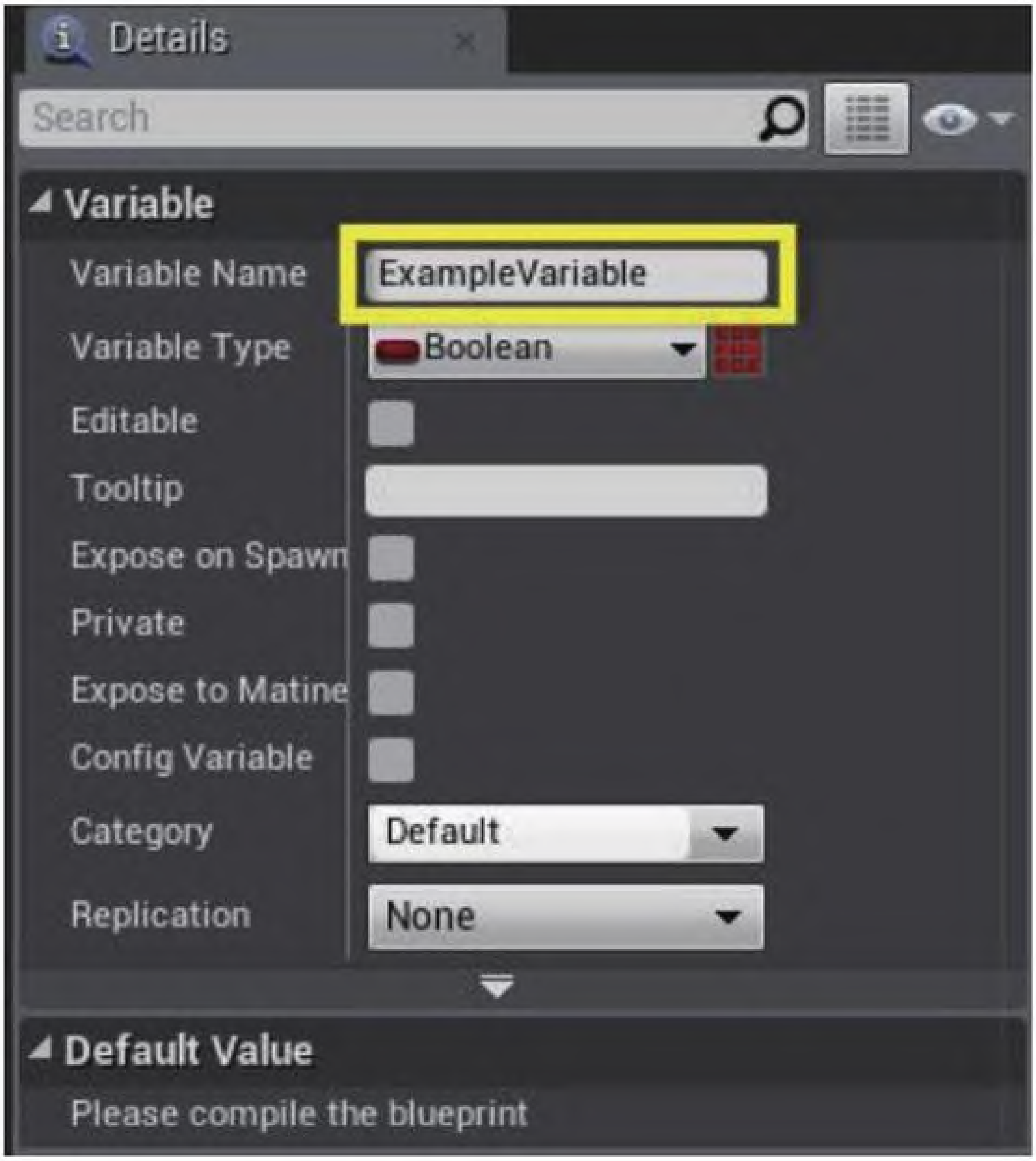

Details
Search
Variable
Variable Name
ExampleVariable
Variable Type
Boolean
Editable
Tooltip
Expose on Spawn
Private
Expose to Matine
Config Variable
Category
Default
Replication
None
Default Value
Please compile the blueprint

图3-20

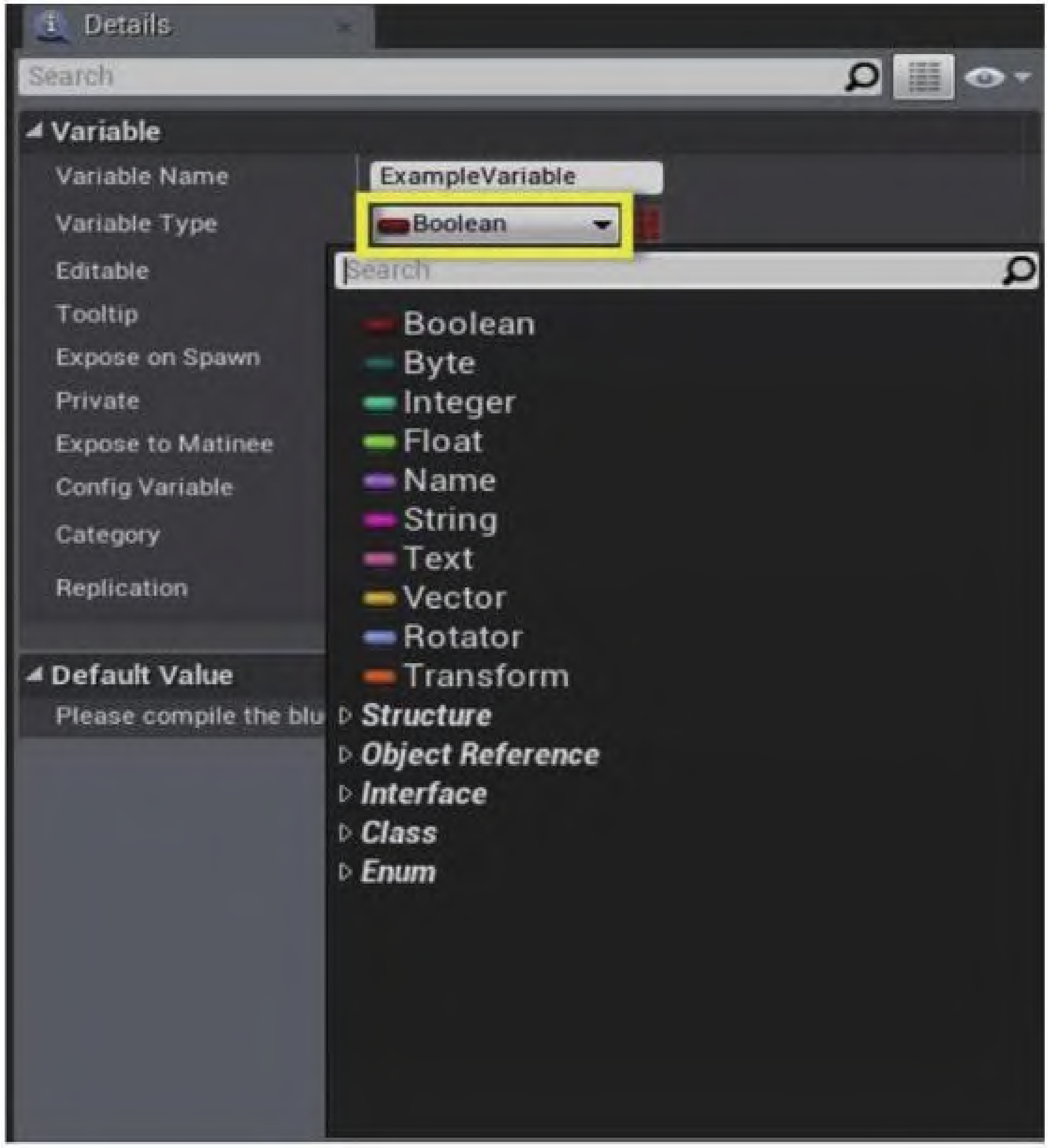



图3-21

如果你正在创建这种变量类型的数组，请单击Variable Type（变量类型）下拉框旁边的Array（数组）按钮，如图3-23所示。

### 2. 提升变量

你可以把一个节点上的任何输入或输出引脚提升为变量，除非该引脚是无类型的数组引脚，如图3-24所示。

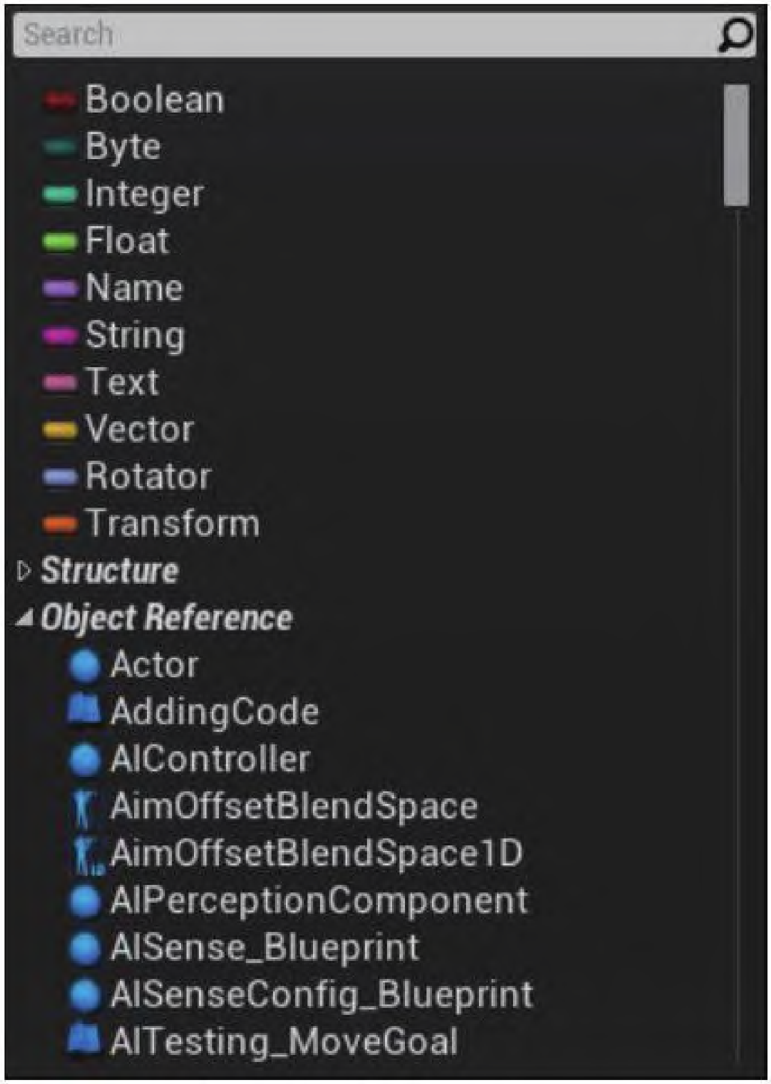

Search
Boolean
Byte
Integer
Float
Name
String
Text
Vector
Rotator
Transform
Structure
Object Reference
Actor
AddingCode
AIController
AimOffsetBlendSpace
AimOffsetBlendSpace1D
AIPerceptionComponent
AISense_Blueprint
AISenseConfig_Blueprint
AITesting_MoveGoal

图3-22

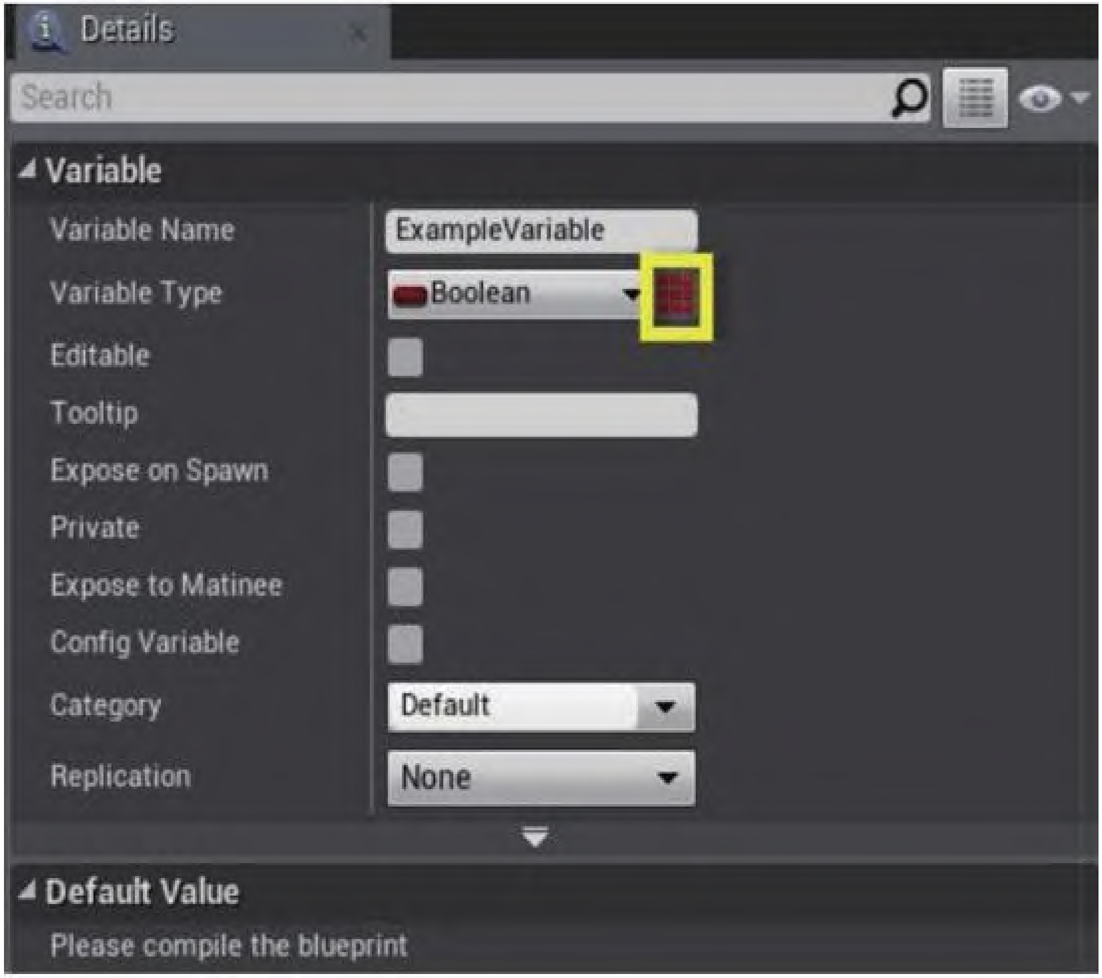

Details
Search
Variable
Variable Name
ExampleVariable
Variable Type
Boolean
Editable
Tooltip
Expose on Spawn
Private
Expose to Matinee
Config Variable
Category
Default
Replication
None
Default Value
Please compile the blueprint

图3-23

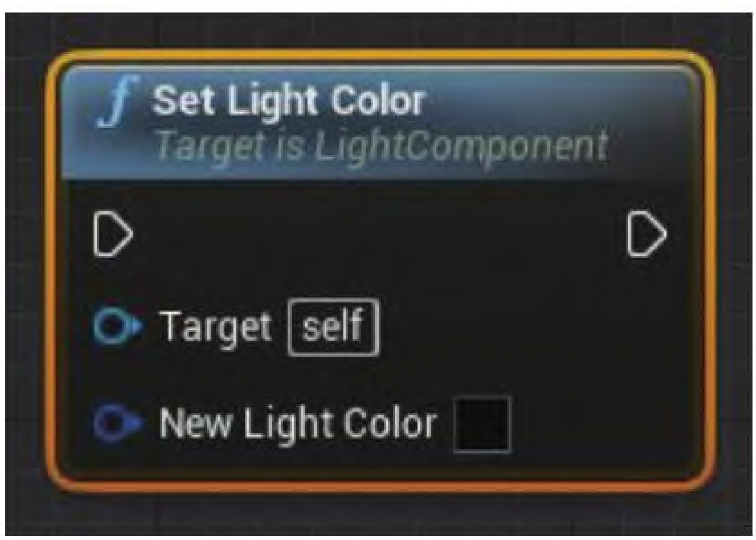

Set Light Color
Target is LightComponent
Target
self
New Light Color

图3-24

Step 1　右击你想提升为变量的引脚，然后从弹出的菜单中选择Promote to Variable（提升为变量），如图3-25所示。

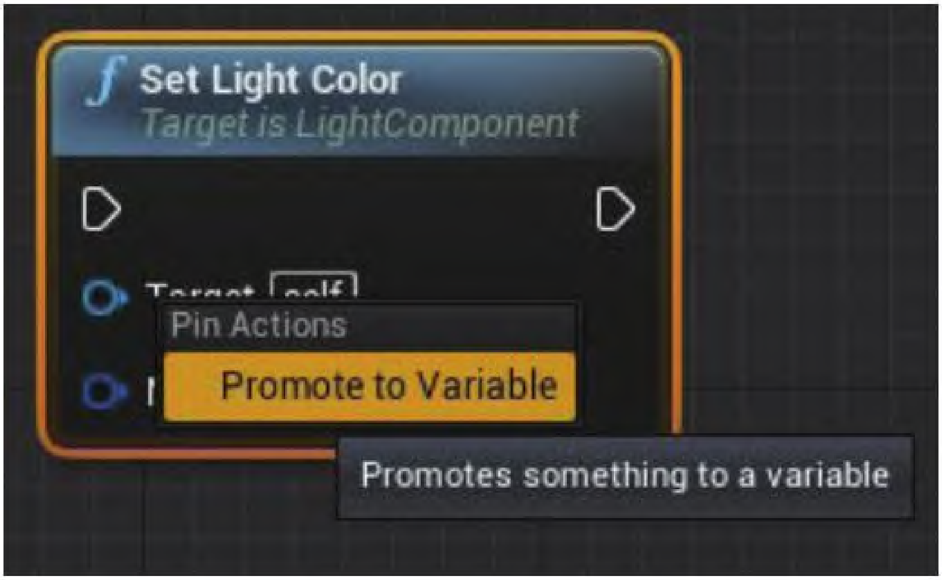



图3-25

Step 2　在新变量名称菜单中输入你的新变量的名称，如图3-26所示。

图3-26

图表上将出现那个新变量的引用。将一个输入引脚提升为变量将创建一个Get节点，将一个输出引脚提升为变量将创建一个Set节点，如图3-27所示。

图3-27

你的新变量将列出在我的蓝图选卡中，在那里你可以选择它并在Details（详细信息）选卡中编辑它的详细信息。

## 3.1.3 组件

组件是一种特殊类型的对象，用作Actor中的一个子对象。组件一般用于需要简单地切换的部件的地方，以便改变具有该组件的Actor的某个特定方面的行为或功能。比如，一辆汽车的控制及运动和飞机是有很大差别的，而飞机的控制和运动又和船有很大区别，以此类推；然而，所有这些都是交通工具，它们存在一些共性。通过使用一个组件来处理这些控制及运动，可以很轻松地使同一交通工具的行为变得像任何一种特定类型的交通工具。

### 3.1.3.1 组件窗口

Components（组件）面板允许组件一创建就添加到蓝图上。这提供了一种方法，使得可以通过CapsuleComponents、BoxComponents或SphereComponents来添加碰撞几何体，可以通过StaticMeshComponents或SkeletalMeshComponents的形式添加渲染几何体，通过使用MovementComponents控制运动等。你还可以将添加到Components（组件）列表中的组件，分配给这个蓝图或其他蓝图中用于访问这些组件的实例变量，如图3-28所示。

图3-28

#### 3.1.3.2 添加组件

要想在组件列表中向蓝图添加一个组件。

**Step 1** 从该下拉列表中选择你想添加的组件类型，比如一个“CameraComponent（相机组件）”，如图3-29所示。

**Step 2** 按下 Add Component 按钮来添加此新组件，如图3-30所示。

Components
+ Add Component
Search Components
New C++ Component...
New Blueprint Script Component...
Common
Audio
Particle System
Point Light
Scene
Skeletal Mesh
Static Mesh
Sphere
Cube
AI
AIPerception
Pawn Noise Emitter
Pawn Sensing
Audio
Audio
Basic Shapes
Cube
Sphere
Cylinder
Cone

图3-29

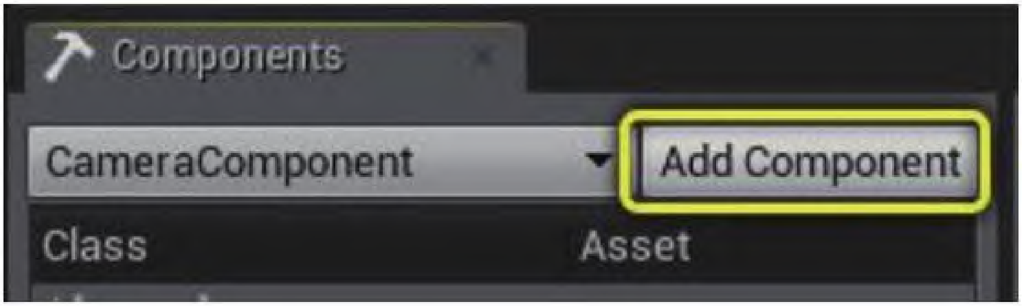



图3-30

要想通过拖动并放置方式来添加一个组件：

组件也可以这样进行添加，即通过从内容浏览器中拖动StaticMesh、SoundCue、SkeletalMesh和ParticleSystem资源并将其放置到组件列表中，如图3-31所示。

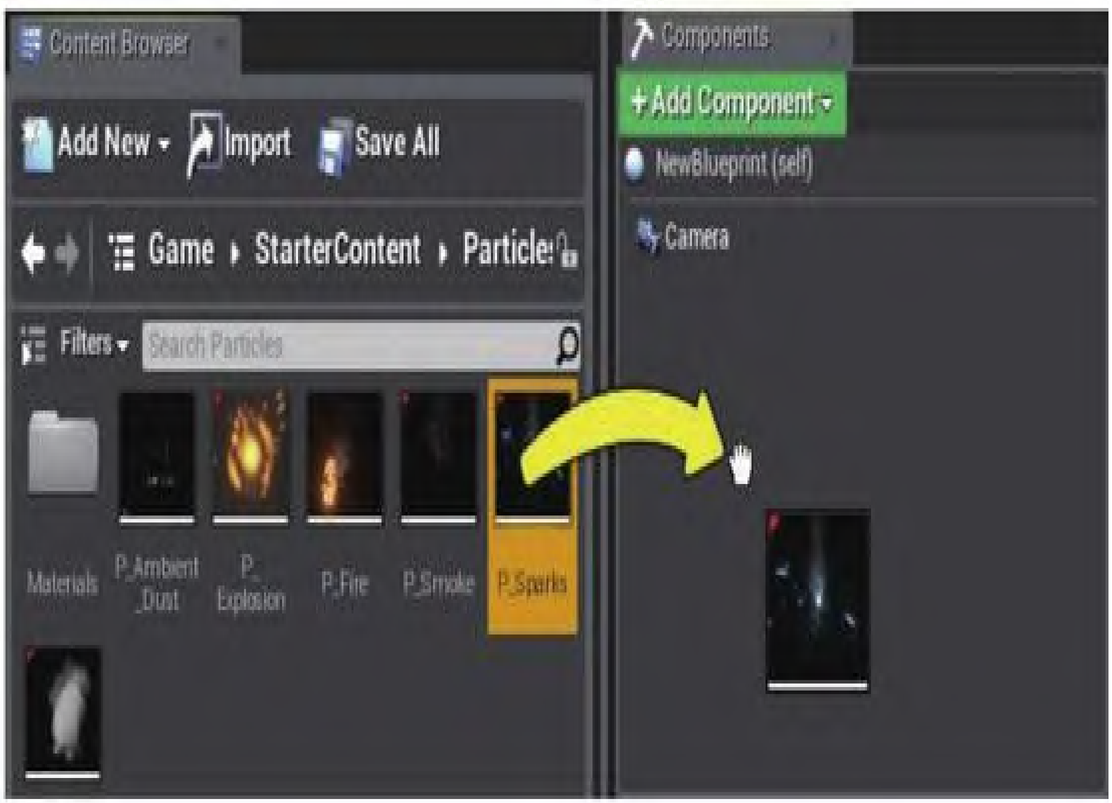



图3-31

### 3.1.3.3　删除组件

要想从Components（组件）列表中删除一个组件，仅需简单地右击该组件的名称并选择Delete（删除）即可，如图3-32所示。

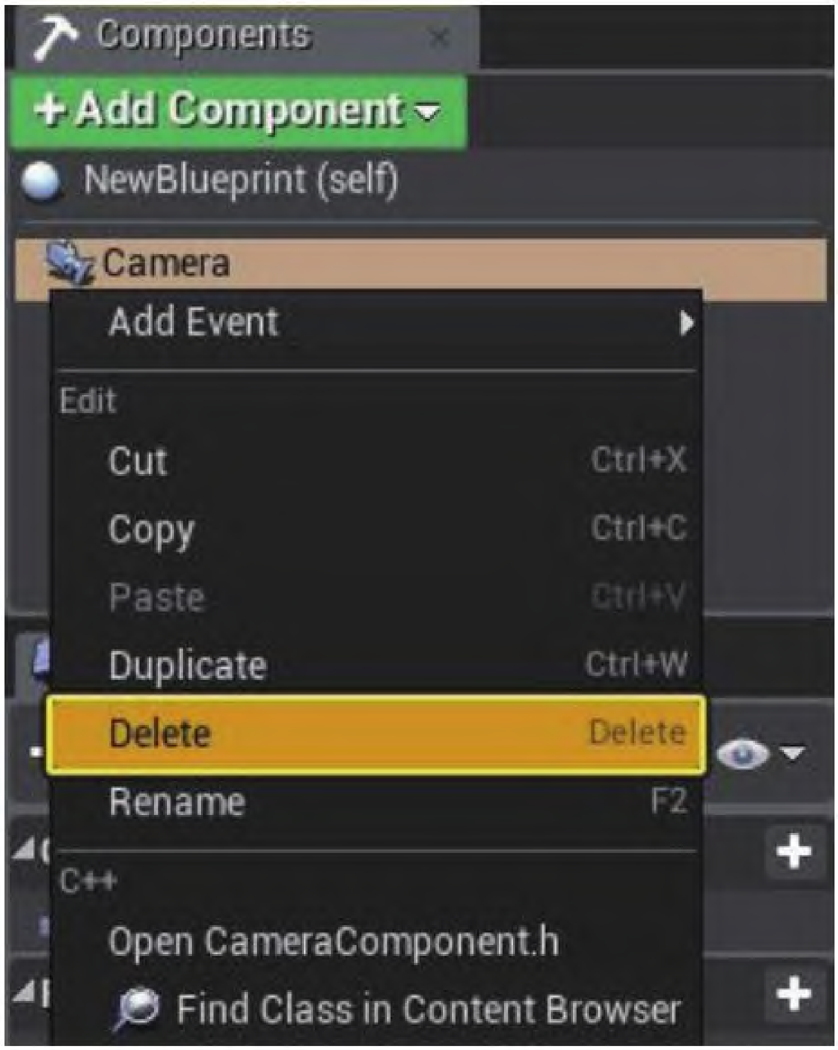



图3-32

### 3.1.3.4 组件资源

#### 1. 把一个资源分配给组件列表中的一个组件

**Step 1** 在内容浏览器中，选择要分配的资源。该资源类型必须和组件类型相兼容，如图3-33所示。

**Step 2** 在组件面板中，选择你想将该资源赋予的组件。该组件的详细信息将会出现在详细信息面板中。

按下详细信息面板Asset（资源）部分中的 按钮。这样该资源就分配给了该组件，如图3-34所示。

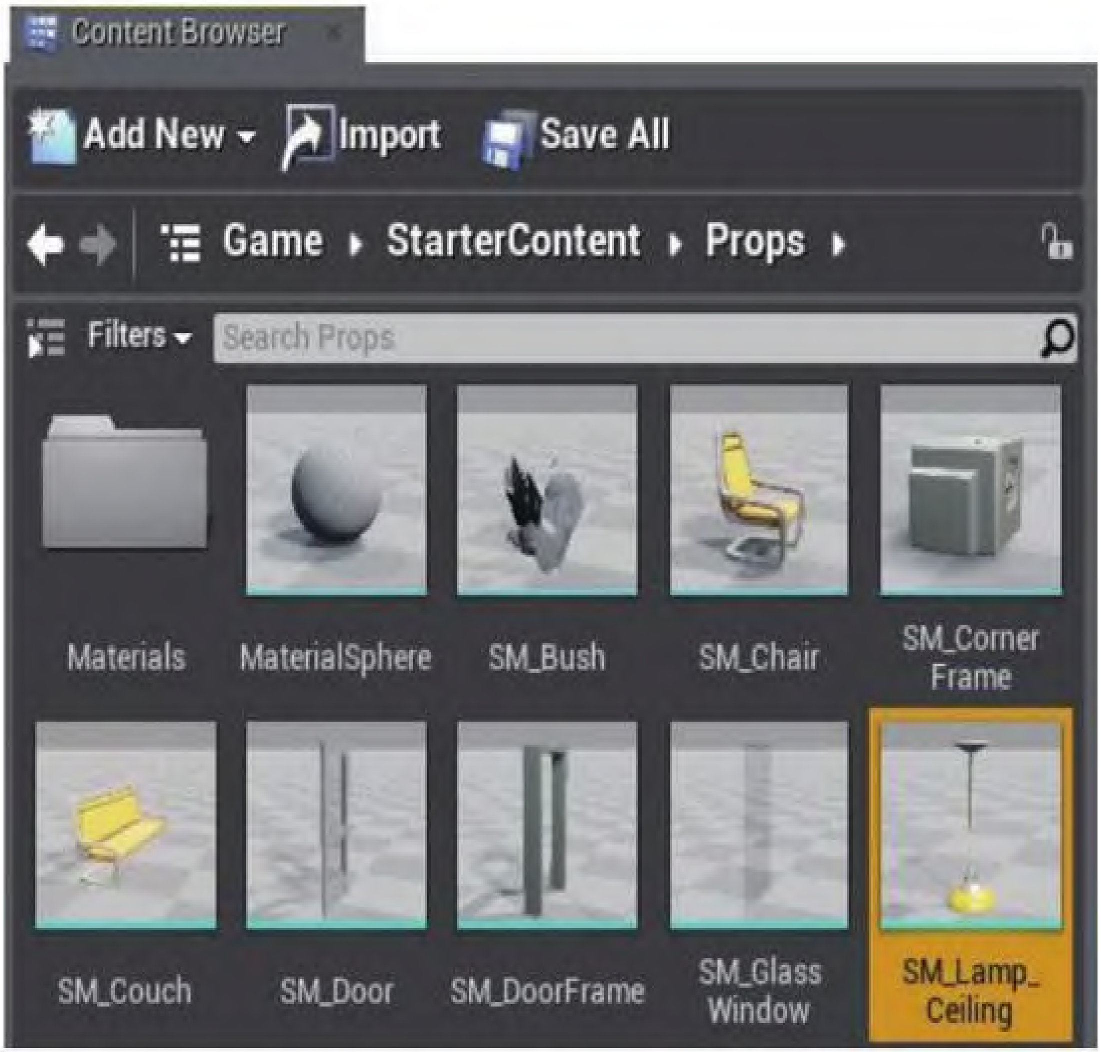



图3-33

图3-34

### 2. 清除已分配给组件的资源

**Step 1** 单击详细信息面板（资源）部分中的Current Asset（当前资源）下拉菜单，如图3-35所示。

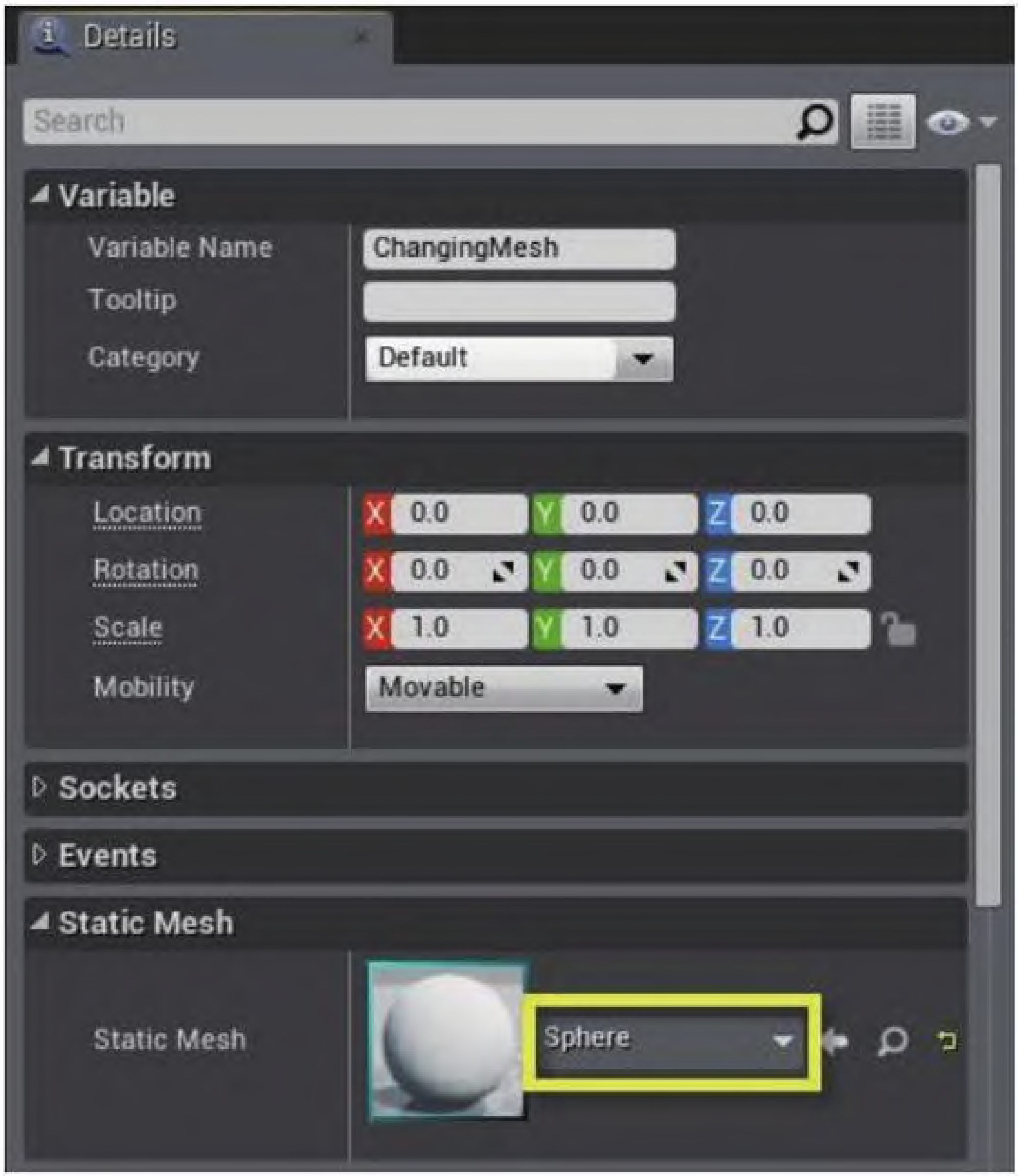



图3-35

**Step 2** 在出现的菜单中选择Current Asset（当前资源）下的Clear（清除）选项，如图3-36所示。

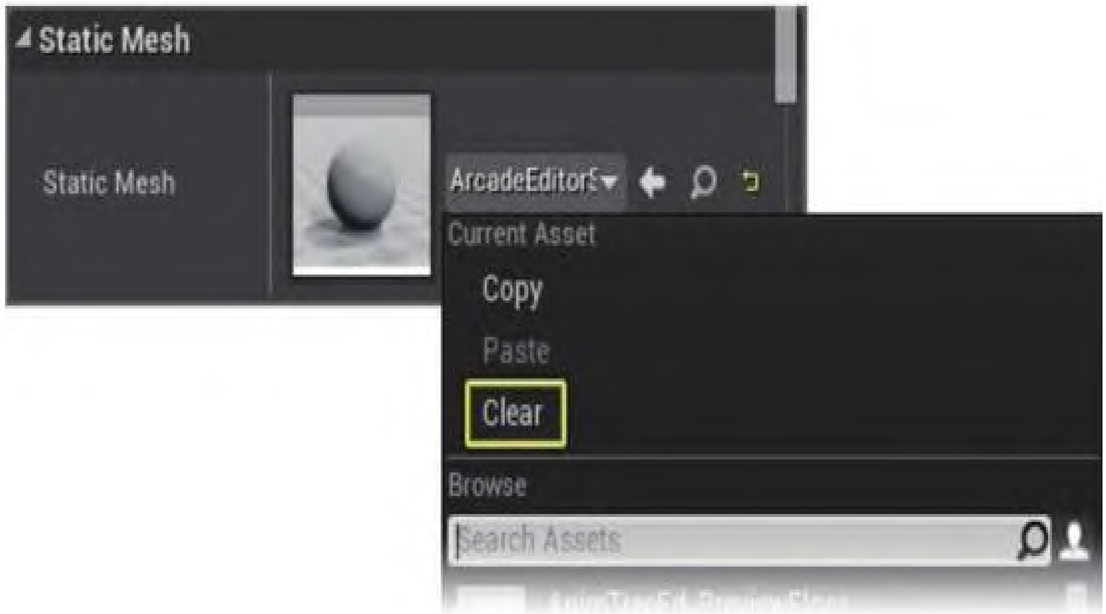



图3-36

该已分配的资源会被清除，如图3-37所示。

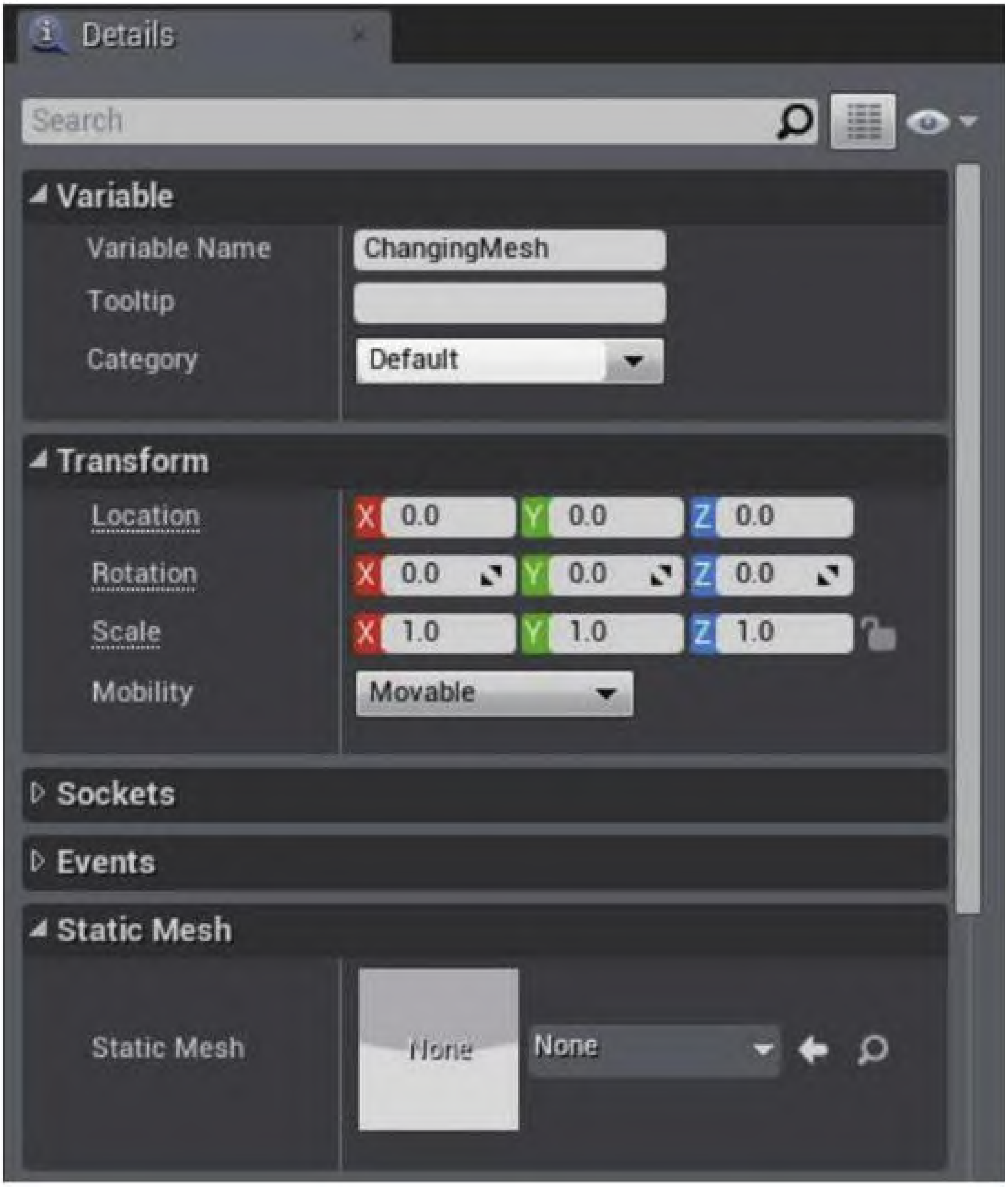



图3-37

### 3. 在内容浏览器中浏览已分配的资源

按下详细信息面板Asset（资源）部分中的 按钮。内容浏览器将会打开并显示已分配的资源，且该资源处于选中状态，如图3-38和图3-39所示。

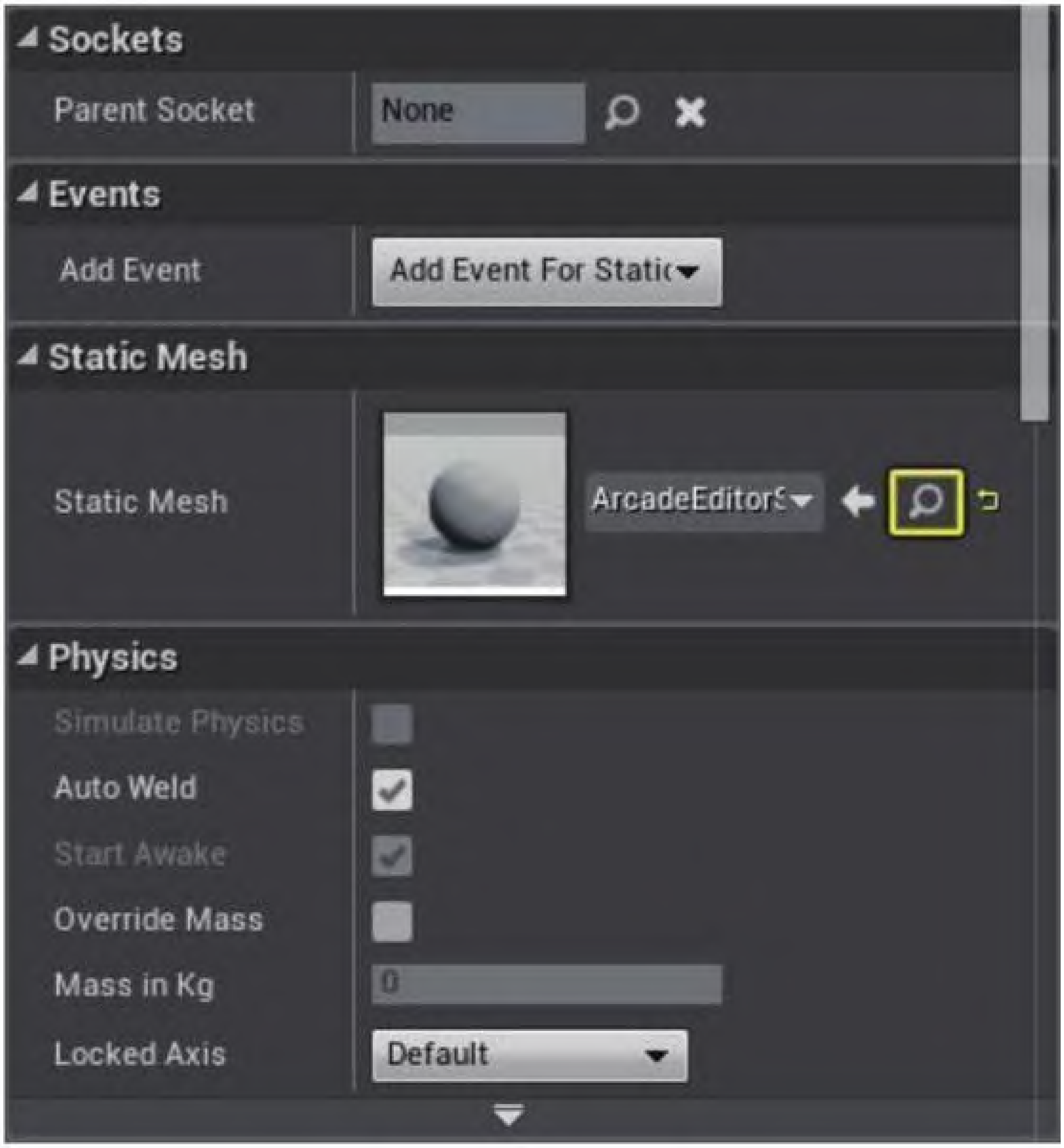

Sockets
Parent Socket
None
Events
Add Event
Add Event For Static
Static Mesh
Static Mesh
ArcadeEditorS
Physics
Simulate Physics
Auto Weld
Start Awake
Override Mass
Mass in Kg
0
Locked Axis
Default

图3-38

## 3.1.3.5 重命名组件实例变量

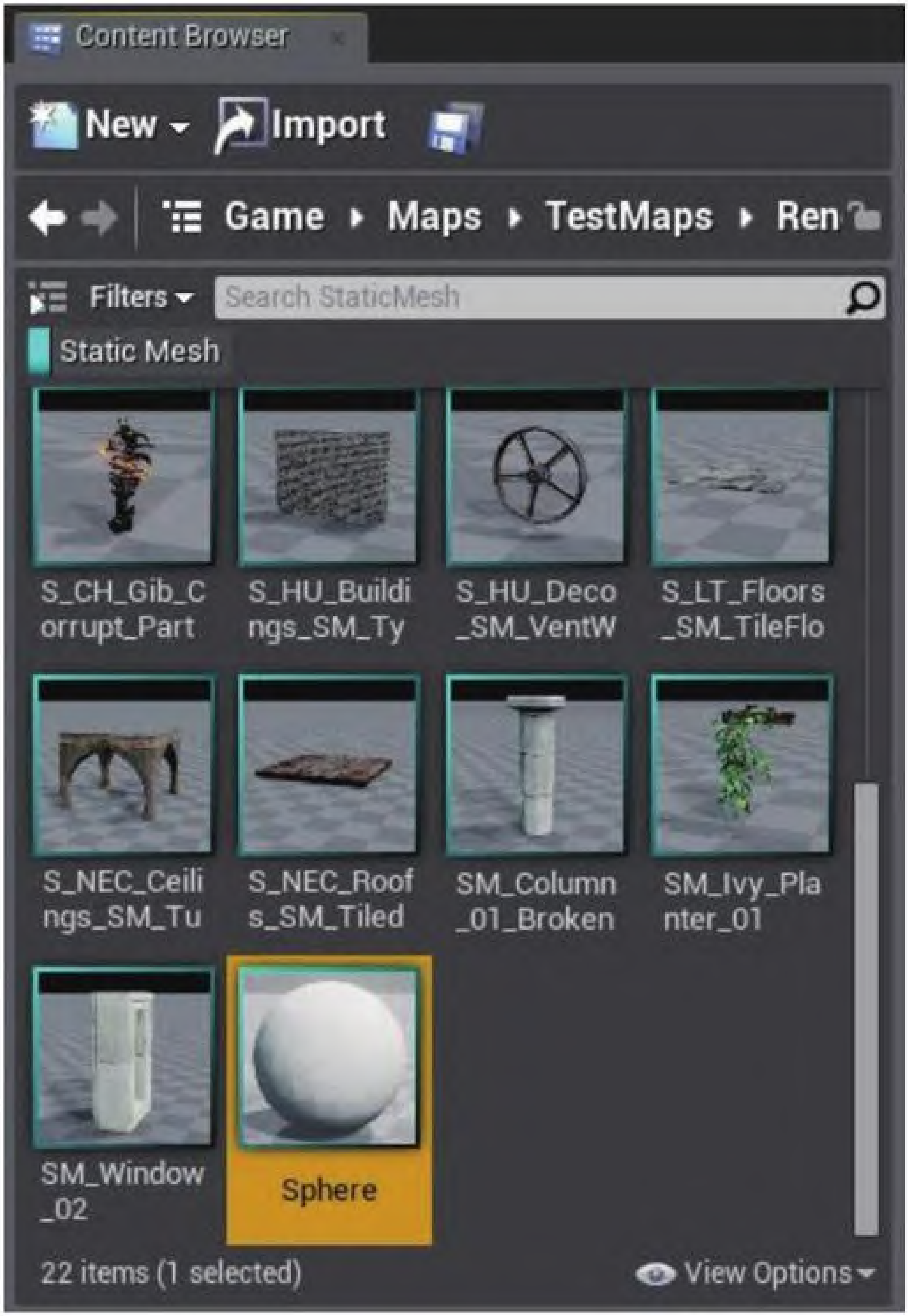



图3-39

### 3.1.3.5 重命名组件实例变量

组件列表中创建的组件将会具有一个基于其类型自动生成的变量名称。

要想修改这些变量的名称：

**Step 1**　在组件列表中选择该变量，其详细信息将出现在Details（详细信息）面板中，如图3-40所示。

**Step 2**　在详细信息面板的组件部分的Variable Name（变量名称）文本域中，为该变量输入一个新名称，如图3-41所示。

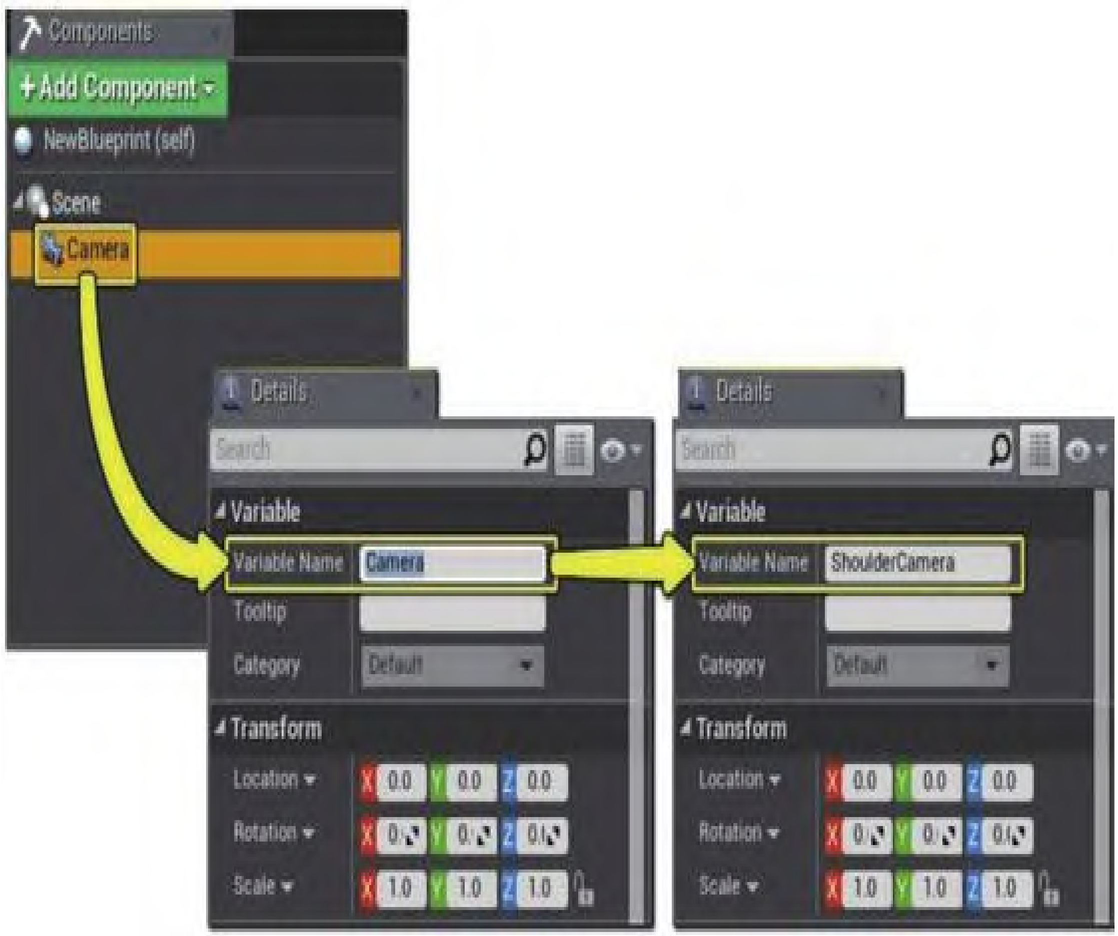



图3-40

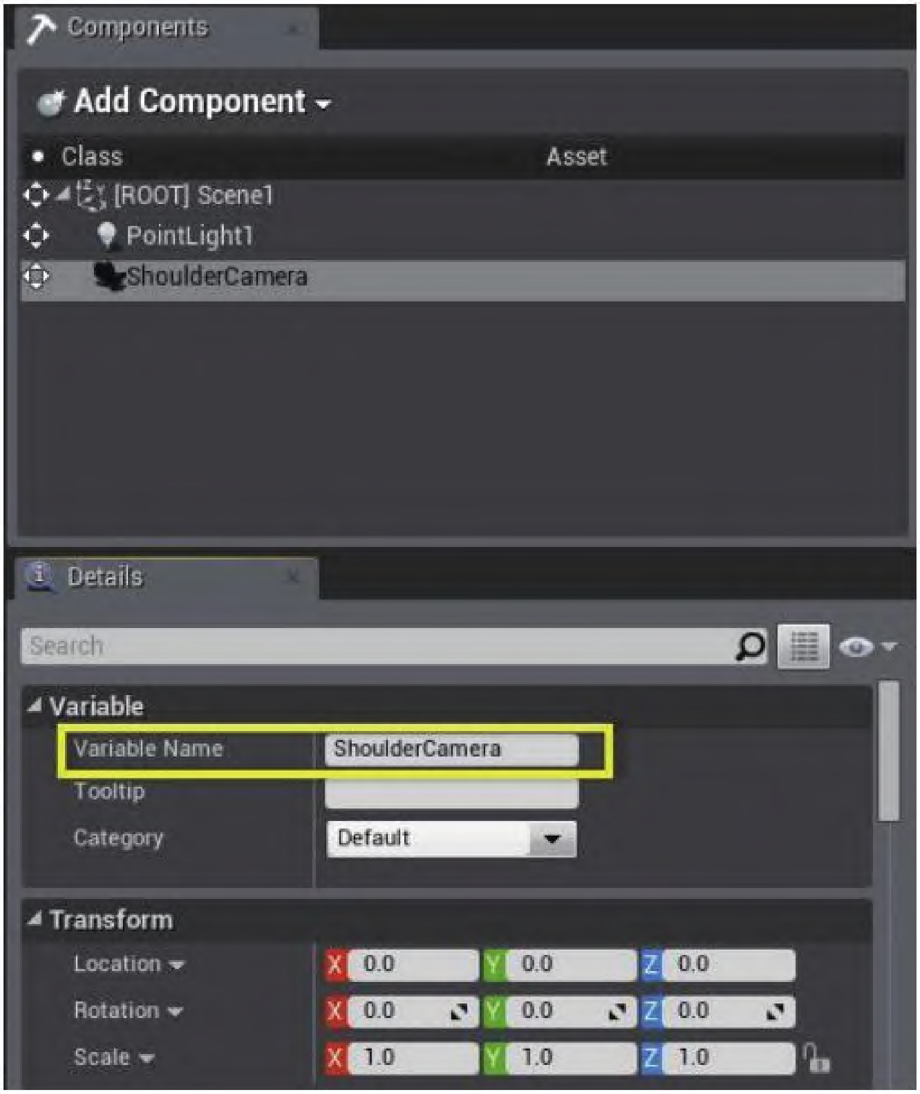



图3-41

### 3.1.3.6　组件事件和函数

根据组件类型的不同，你可以基于它们在事件图表中添加事件和/或函数。比如，另一个Actor（比如一个Pawn）穿过Box Component时，该Box Component可以有一个Overlap事件。以这种方式创建的任何事件或函数都是针对那个特定函数的，不必测试来确认它所涉及的组件。

创组件事件非常简单，方法如下：

**Step 1**　创建一个可以为其创建事件的组件。在这个示例中，我们使用一个BoxComponent（盒体组件），如图3-42所示。

**Step 2**　为你的组件提供一个名称，这会为其创建一个变量，如图3-43所示。

**Step 3**　在Variables（变量）选卡中，选择该新变量，如图3-44所示。

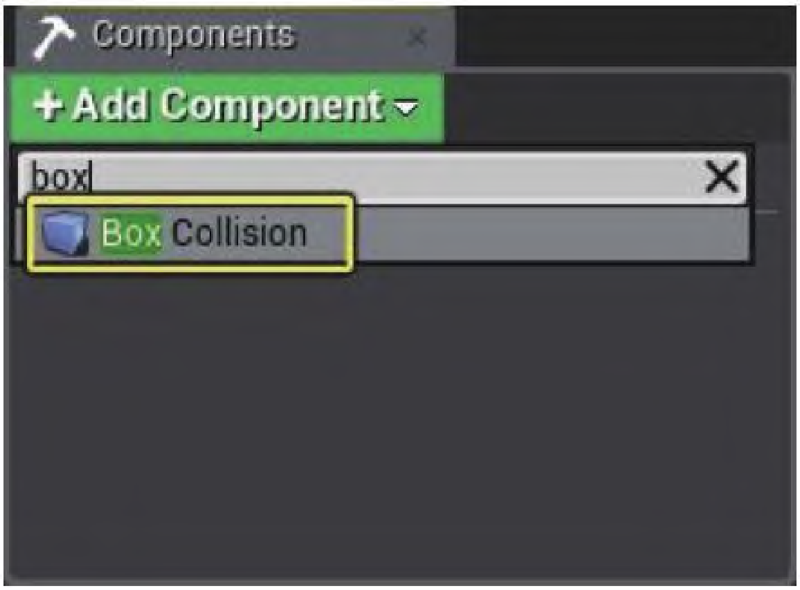



图3-42

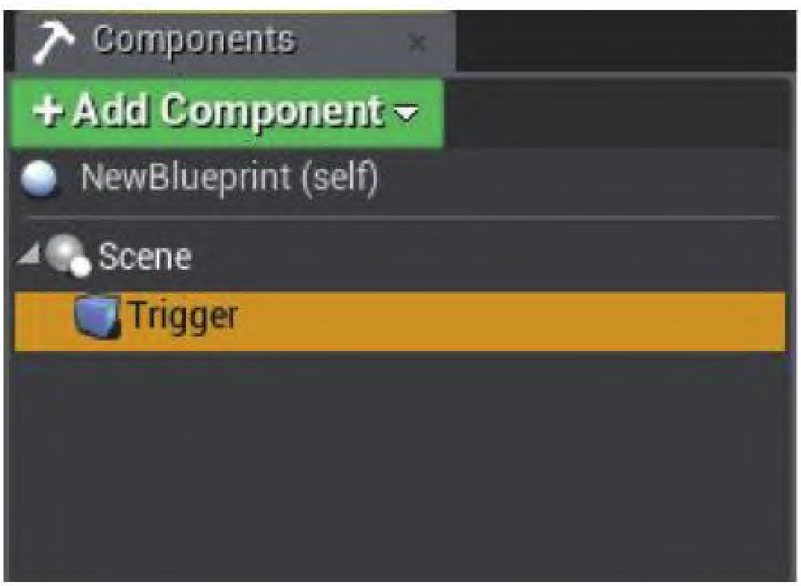



图3-43

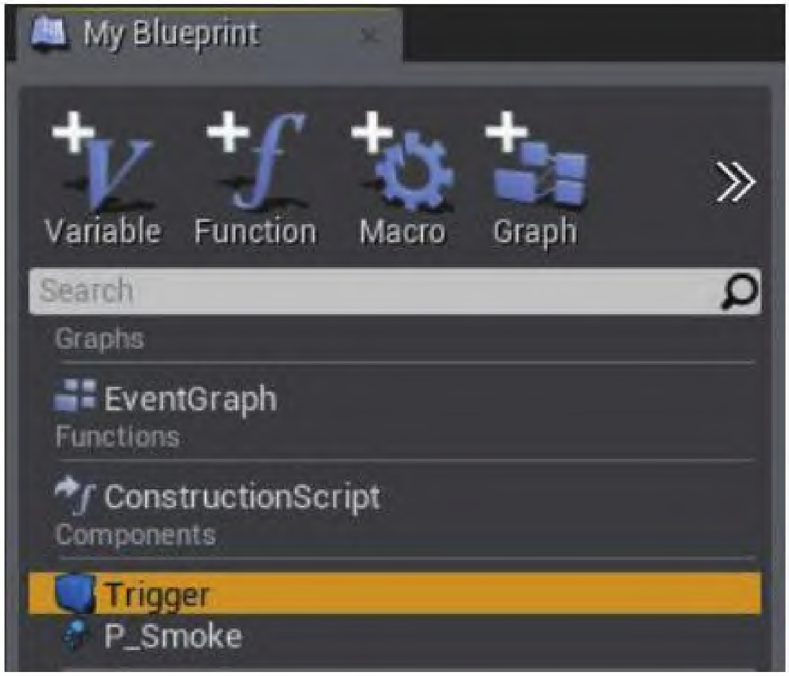



图3-44

**Step 4**　在图表选卡中，右击弹出关联菜单。如果该组件有任何相关联的事件或函数，都会在顶部列出这些事件或函数，如图3-45所示。

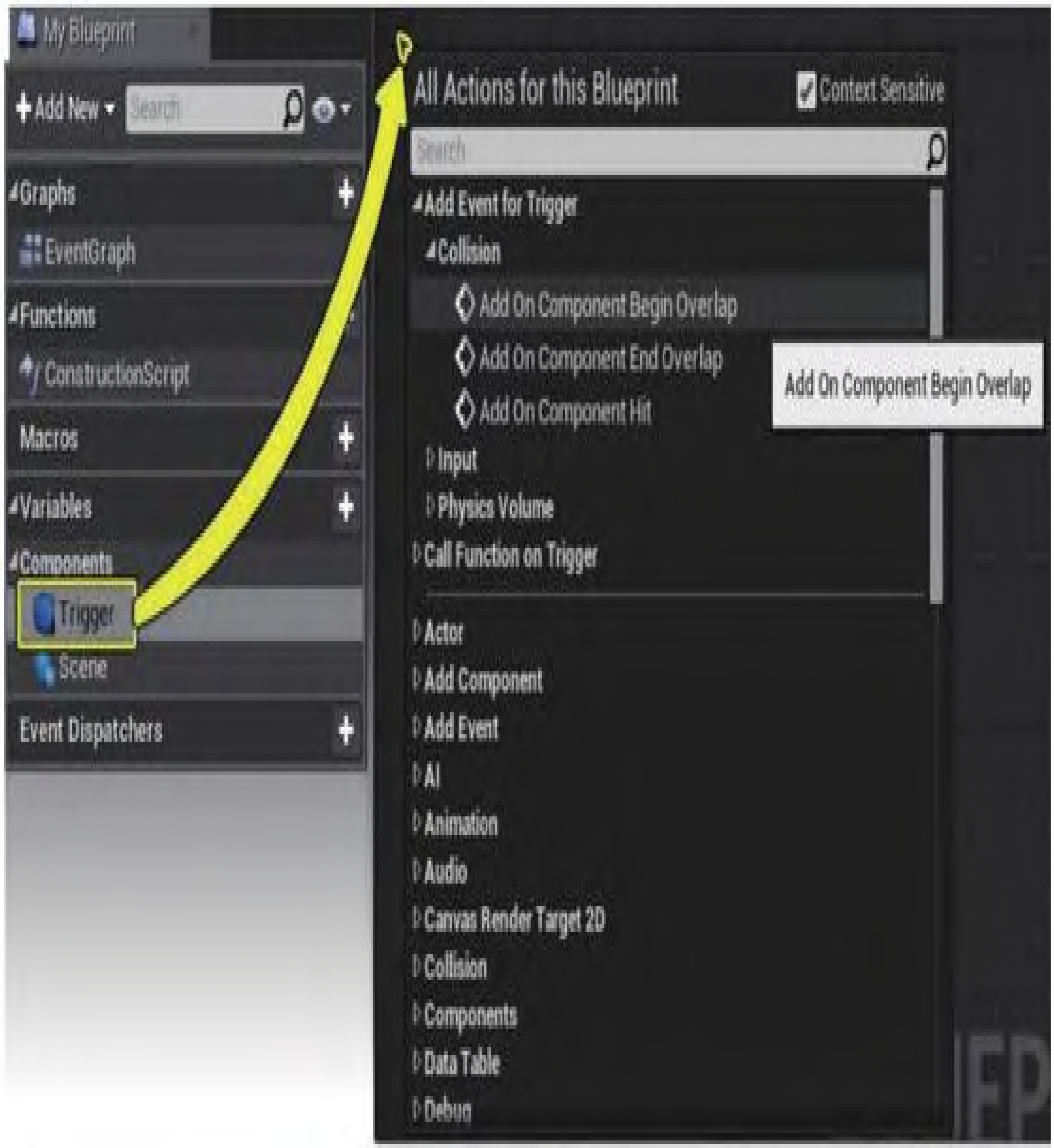



图3-45

在这个示例中，我们添加了一个Overlap事件，每当有Actor穿过这个Box Component体积时都会调用该事件。

不是所有的组件都有相关联的事件。比如，一个Point Light Component（点光源组件）仅具有函数。

### 3.1.4　事件图表

蓝图中的EventGraph（事件图表）包含一个节点图表，该图表使用事件和函数调用来执行动作，以便对和该蓝图相关的游戏事件做出反应。事件图表通常用于给蓝图的所有实例添加功能。这里也是设置交互性和动态反应的地方。比如，一个光照蓝图可以通过关闭其LightComponent及改变其网格物体使用的材质来对于伤害事件做出反应。这将会自动地为该光照蓝图的所有实例提供这种行为，如图3-46所示。

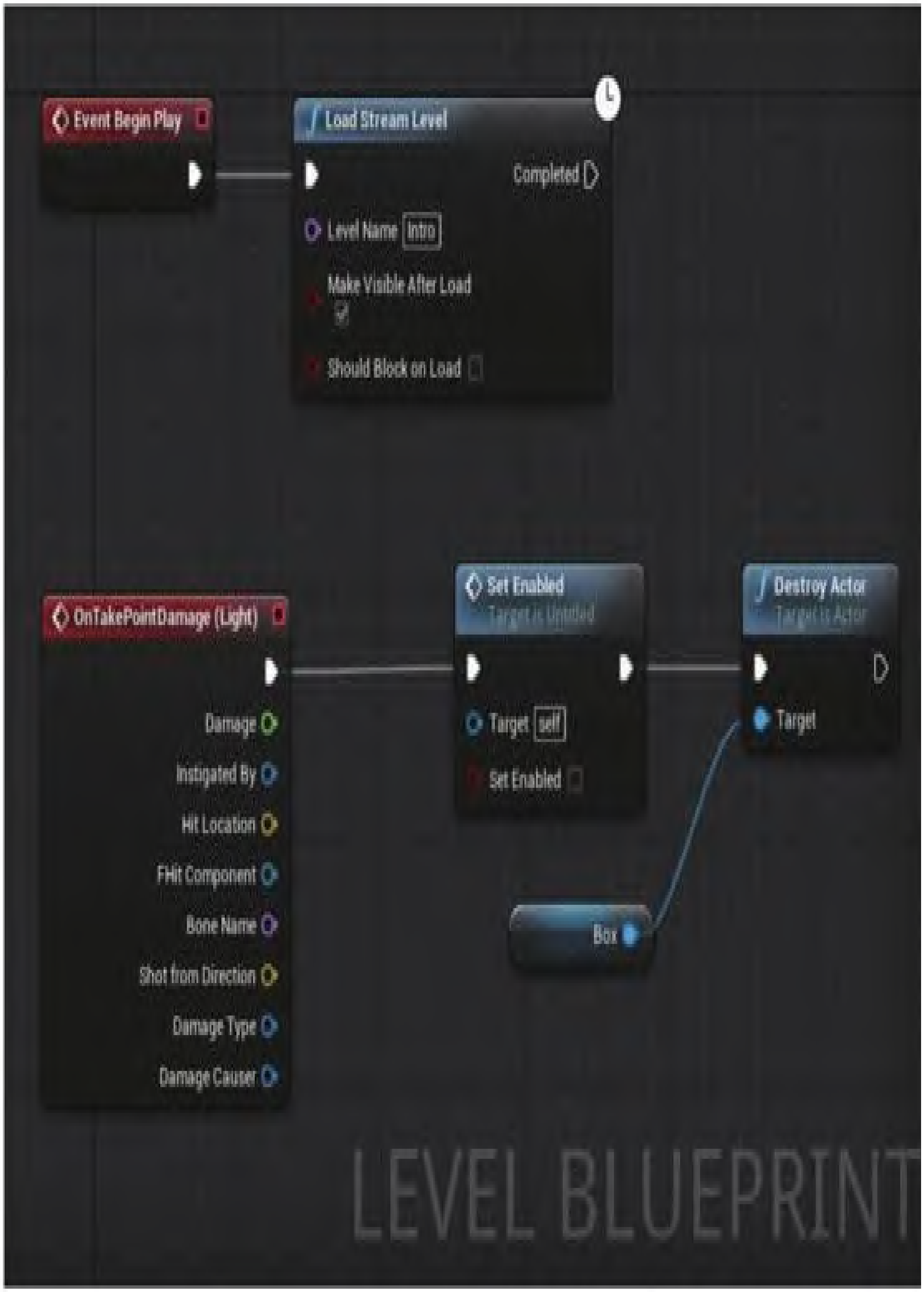

Event Begin Play
Load Stream Level
Completed
Level Name
Intro
Make Visible After Load
Should Block on Load
OnTakePointDamage (Light)
Damage
Instigated By
Hit Location
FHit Component
Bone Name
Shot from Direction
Damage Type
Damage Causer
Set Enabled
Target
self
Set Enabled
Destroy Actor
Target is Actor
Target
Box
LEVEL BLUEPRINT

图3-46

关卡蓝图中的EventGraph（事件图表）包含一个节点图表，该图表使用事件和函数调用来执行动作，以便对游戏事件做出反应。这个事件图表用于从总体上处理关卡中的事件，及给世界中特定的Actor和蓝图实例添加功能。

无论哪种情况，应用EventGraph（事件图表）都要通过添加一个或多个事件来作为入口点，然后连接Function Calls（函数调用）、Flow Control（流程控制）节点及变量来执行期望的动作。

## 3.1.5　图表

Graph（图表）是由一个个彼此相连的节点构成的网络，以便可以定义该网络的执行流程。图表是在蓝图中实现功能的基础。根据蓝图类型不同，每个蓝图可以包含一个或多个图表，图表中定义了蓝图的某个特定方面功能的实现。蓝图中的每个图表还可以包含子图表，这些子图表实际上是一组节点浓缩构成的独立图表，主要用于图表的组织管理。有一些专门的图表类型，它们具有一些独特的属性，但是添加变量引用、添加及连接节点、及调试的关键原则适用于所有图表类型。

### 3.1.5.1　图表类型概述

**1.事件图表**

EventGraphs（事件图表）是蓝图图表的一种最通用类型。每个新的类蓝图在创建时都包含一个事件图表，当然还可以在其中添加多个事件图表。这些额外的事件图表可以帮助你组织你的蓝图网络。事件图表一般包含该蓝图的游戏行为的节点网络，使用Events（事件）、自定义事件和Input（输入）节点来启动整个事件图表的执行流程，如图3-47所示。

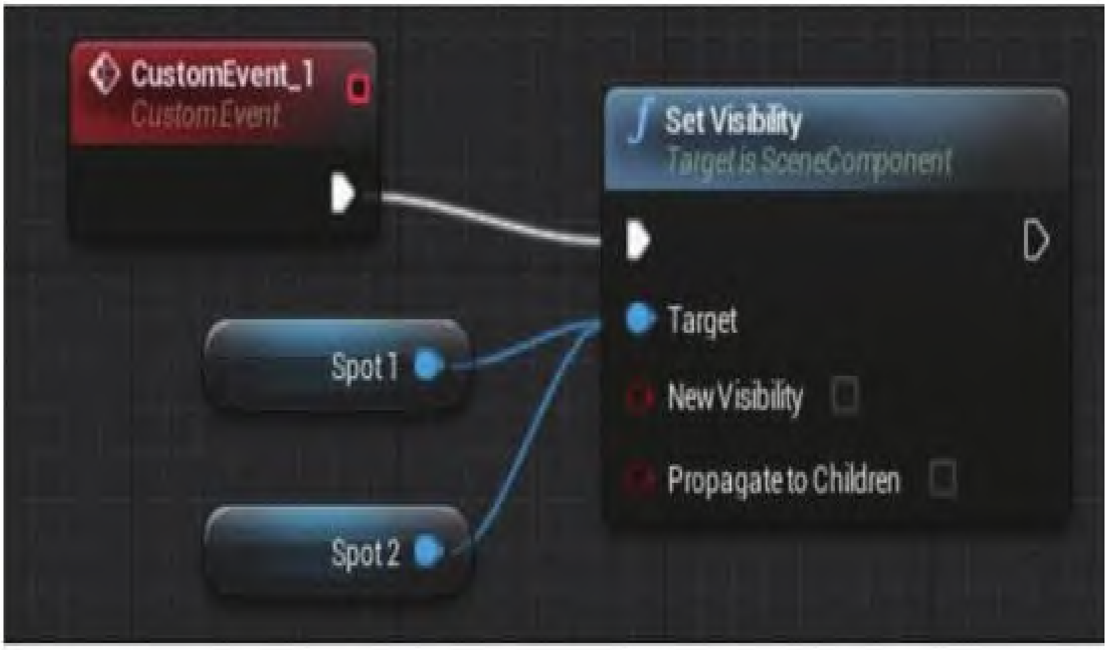



图3-47

## 2. 构造脚本

ConstructionScripts（构造脚本）是Class Blueprints（类蓝图）所特有的，每个类蓝图中仅存在一个ConstructionScript图表。ConstructionScripts对于类蓝图初始化是非常有用的，因为它们恰好在为该类蓝图设置的组件列表之后运行，如图3-48所示。

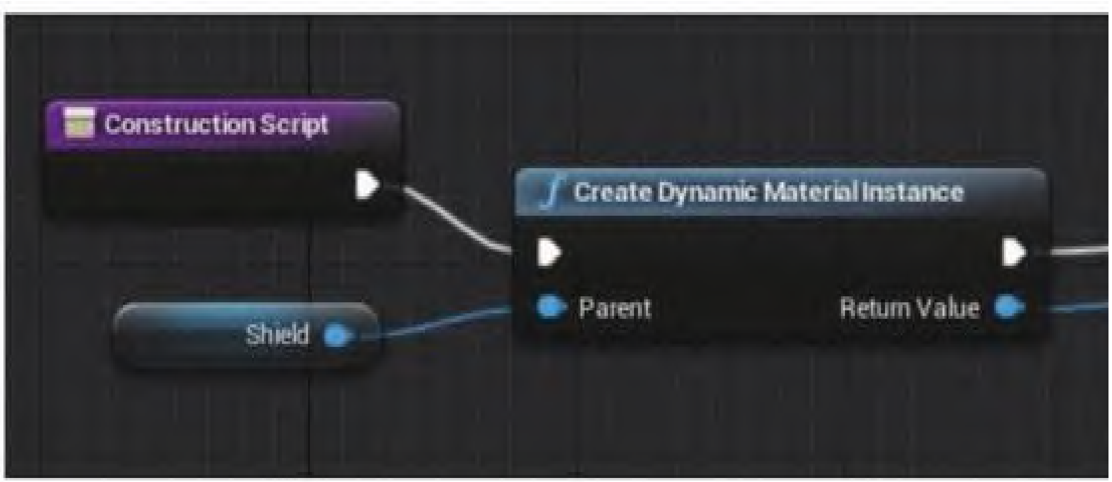



图3-48

## 3. 函数

Functions（函数）是属于特定蓝图的节点图表，该节点图表可以执行或者从蓝图中的另一个图表中进行调用。函数具有一个单一的入口点，由和该函数具有相同名称的一个节点指定，该节点包含了一个单一的执行输出引脚。当从另一个图表调用函数时，会激活该输出执行引脚，促使执行连接网络，如图3-49所示。

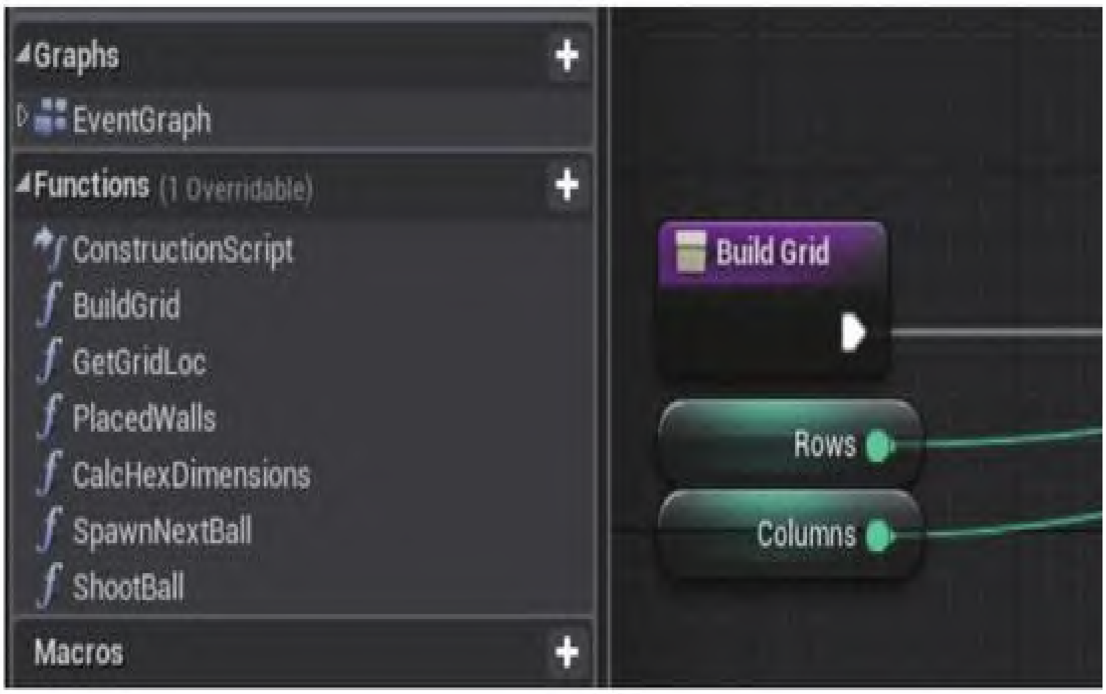



图3-49

## 4. 宏

蓝图宏或宏从本质上讲和合并的节点图表一样。它们具有由通道节点指定的一个入口点和一个出口点。每个通道可以具有多个执行引脚或数据引脚，当在其他蓝图及图表中使用该宏节点时这些引脚会呈现在宏节点上，如图3-50所示。

图3-50

#### 3.1.5.2 应用图表

无论你的图表是Construction Script（构造脚本）、EventGraph（事件图表）、Function（函数）还是Macro（宏），你都可以在蓝图编辑器参考的图表模式中对其进行编辑。基本上，所有的图表都包含由连线连接起来的节点网络。

### 3.1.6 构造脚本

当创建类蓝图时，Construction Script（构造脚本）将在Components（组件）列表之后运行。构造脚本包含一个节点图表，执行该图表使得该类蓝图实例执行初始化操作。这个功能非常强大，因为像在世界中执行追踪、设置网格物体及材质等这样的动作可以用于实现情境关联的设置。比如，一个光源的蓝图可以决定它所放置到的地面的类型，且可以从一组网格物体中选择要使用的正确的网格物体；或者一个篱笆蓝图，它可以在各个方向上进行追踪，来决定篱笆需要有多长才能跨越那个距离，如图3-51所示。

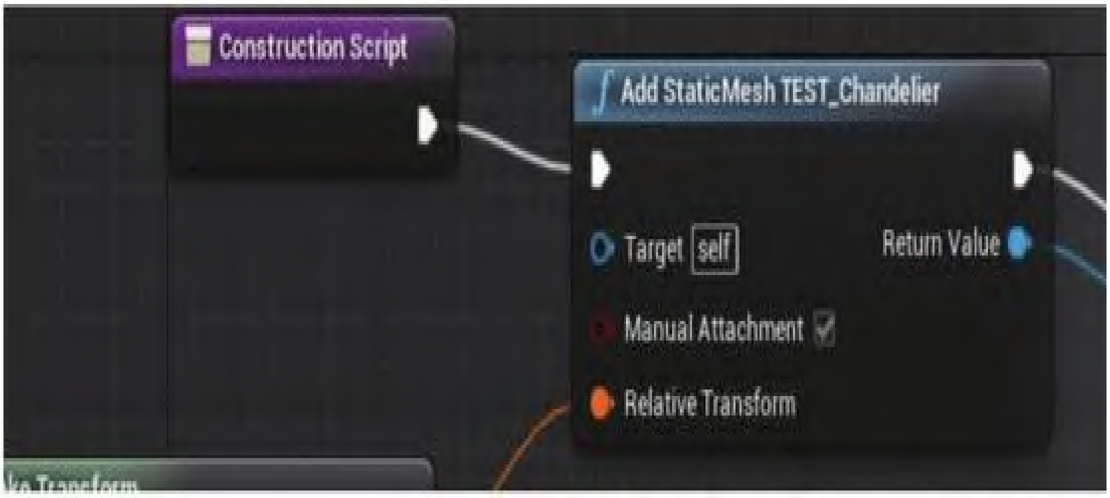



图3-51

仅类蓝图包含Construction Scripts（构造脚本）。关卡蓝图没有构造脚本。

Construction Script（构造脚本）图表的执行入口点是一直存在的ConstructionScript节点，如图3-52所示。

图3-52

3.2

# 用蓝图工作

当你用蓝图工作时，可以对图表进行注释，在项目中所有的蓝图内进行搜索，以及使用断点工具来调试。

## 3.2.1　蓝图编辑器速查表

蓝图内置了很多提高效率的快捷方式，随着你应用编辑器，很多快捷方式自然而然地就会用到，但是还有一些快捷方式则略有隐藏。本文介绍了最常用的操作和快捷方式，使用它们进行操作效率会更高。

图表操作，见下表和图3-53。

| 操　作 | 命　令 |
|---|---|
| 图表操作菜单 | 在图表上右击 |

选择，见下表和图3-54。

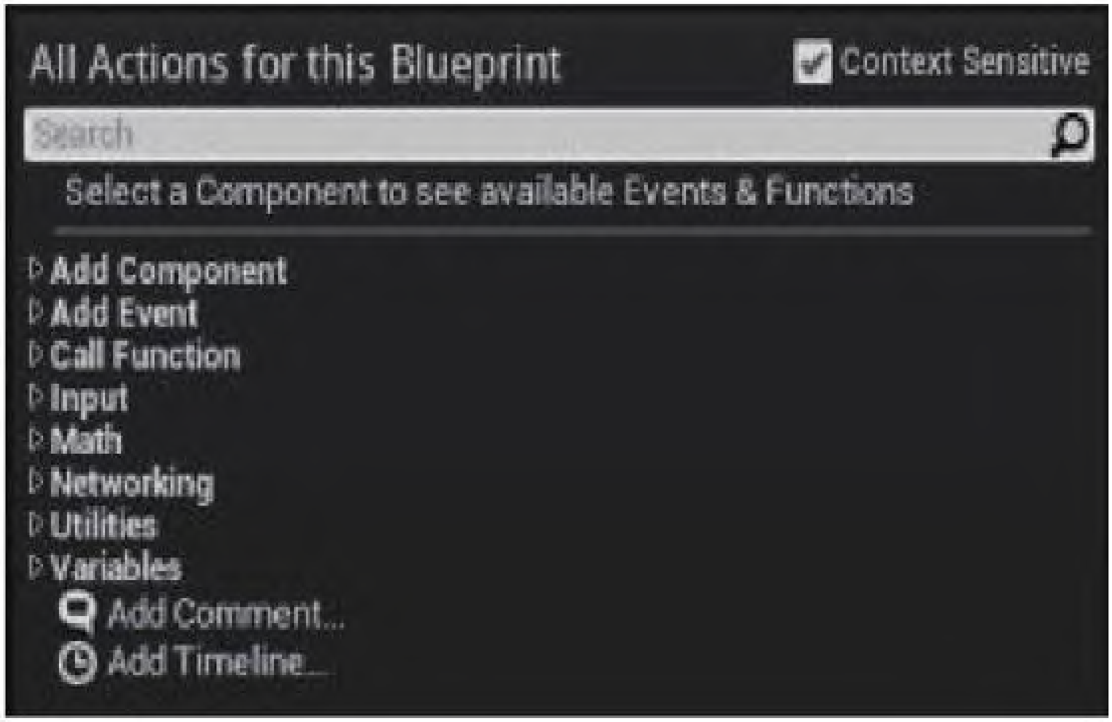

All Actions for this Blueprint
Context Sensitive
Search
Select a Component to see available Events & Functions
Add Component
Add Event
Call Function
Input
Math
Networking
Utilities
Variables
Add Comment...
Add Timeline...

图3-53

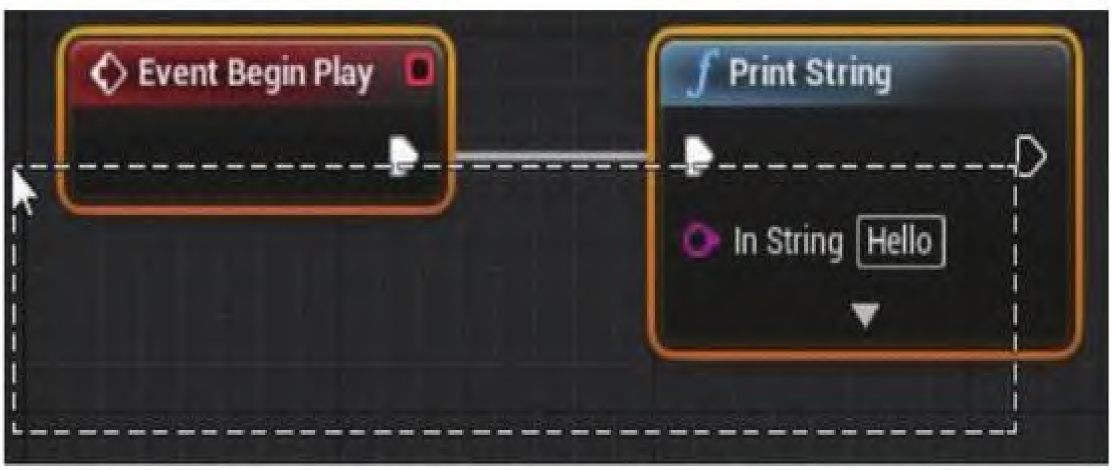

Event Begin Play
Print String
In String
Hello

图3-54

| 动　作 | 命　令 |
| --- | --- |
| 选择节点 | 单击一个节点 |
| 添加到选中项 | Shift+ 单击一个节点 |
| 切换选中状态 | Ctrl+ 单击一个节点 |
| 区域选择（替换） | 鼠标左键拖动 |
| 区域选择（添加） | Shift+ 鼠标组件拖动 |
| 区域选择（删除） | Ctrl+ 鼠标左键拖动 |

导航，见下表和图3-55。

| 动　作 | 命　令 |
| --- | --- |
| 平移图表 | 鼠标右键拖动 |
| 缩放来适应到选中项 | Home（起始键） |
| 放大 / 缩小 | 向上 / 向下滚动鼠标滚轮 |
| 放大 / 缩小 | 按住鼠标左键 + 鼠标右键并拖动 |
| 以超过 1:1 的比例缩放 | Ctrl+ 缩放 |
| 进入到子图表 | 下一页按键 |
| 进入到父项图表 | 上一页按键 |

通用命令，见下表和图3-56。

Event Begin Play
ChildGraph
Collapsed Graph
Navigate to graph
ChildGraph
Pick the graph to enter

图3-55

Event Begin Play
ChildGraph
Collapsed Graph
BLUEPRINT
Find Results
Enter function or event name to find reference
Find In Current Blueprint Only

图3-56

| 动　作 | 命　令 |
| --- | --- |
| 在内容浏览器中查找 | Ctrl+B |
| 保存蓝图 | Ctrl+S |
| 恢复 | Ctrl+Y |
| 取消 | Ctrl+Z |
| 在该蓝图中查找 | Ctrl+F |
| 在任何蓝图中查找 | Ctrl+Shift+F |
| 编译蓝图 | F7 |

变量操作（我的蓝图），见下表和图3-57。

| 操作 | 命令 |
| --- | --- |
| 视情况获得 / 设置变量 | 鼠标左键拖动连接到兼容引脚上 |
| 获得 / 设置变量（通过菜单） | 鼠标左键拖动到图表中 |
| 获得变量 | Ctrl+ 鼠标左键拖动到图表中 |
| 设置变量 | Alt+ 鼠标左键拖动到图表中 |
| 修改现有节点 | 鼠标左键拖动到 Get/Set 的边缘处 |
| 修改类目或重新排序 | 在我的蓝图中使用鼠标左键拖动 |

节点操作，见下表和图3-58。

Event Begin Play
SET
Get Float
Set Float
Make NewGetFloat = GetFloat
BLUEPRINT

图3-57

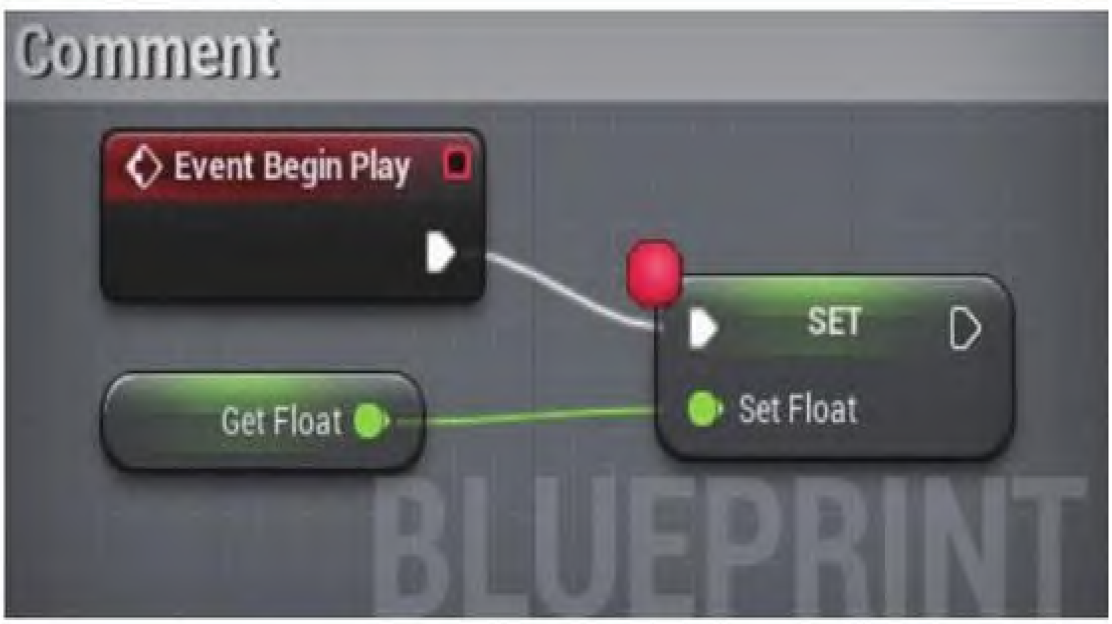

Comment
Event Begin Play
SET
Get Float
Set Float
BLUEPRINT

图3-58

| 操 作 | 命 令 |
|---|---|
| 节点相关的关联菜单 | 右击一个节点 |
| 跳转到相关的节点/图表 | 在一个节点上双击 |
| 移动节点 | 鼠标左键拖动节点 |
| 移动选中的节点 | 方向键 |
| 删除选中的节点 | 删除键 |
| 重命名节点/编辑注释 | 单击标题 |
| 重命名节点/编辑注释 | F2 |
| 切换断点 | F9 |
| 清除所有断点 | Ctrl+Shift+F9 |
| 剪切选中项 | Ctrl+X |
| 复制选中项 | Ctrl+C |
| 粘贴节点 | Ctrl+V |
| 克隆选中项 | Ctrl+W |
| 围绕选中项添加注释 | C |

引脚操作，见下表和图3-59。

| 操　作 | 命　令 |
| --- | --- |
| 引脚相关的关联菜单 | 右击引脚 |
| 着重显示连接线 | 鼠标悬停到引脚上 |
| 连接到另一个引脚 | 左击 + 拖动到那个引脚 |
| 引脚的过滤后的操作菜单 | 左击 + 拖动到图表 |
| 断开所有连接 | Alt+ 左击引脚 |
| 移动所有连接 | Ctrl+ 鼠标左键拖动到引脚 |

创建相关的快捷方式，见下表和图3-60。

图3-59

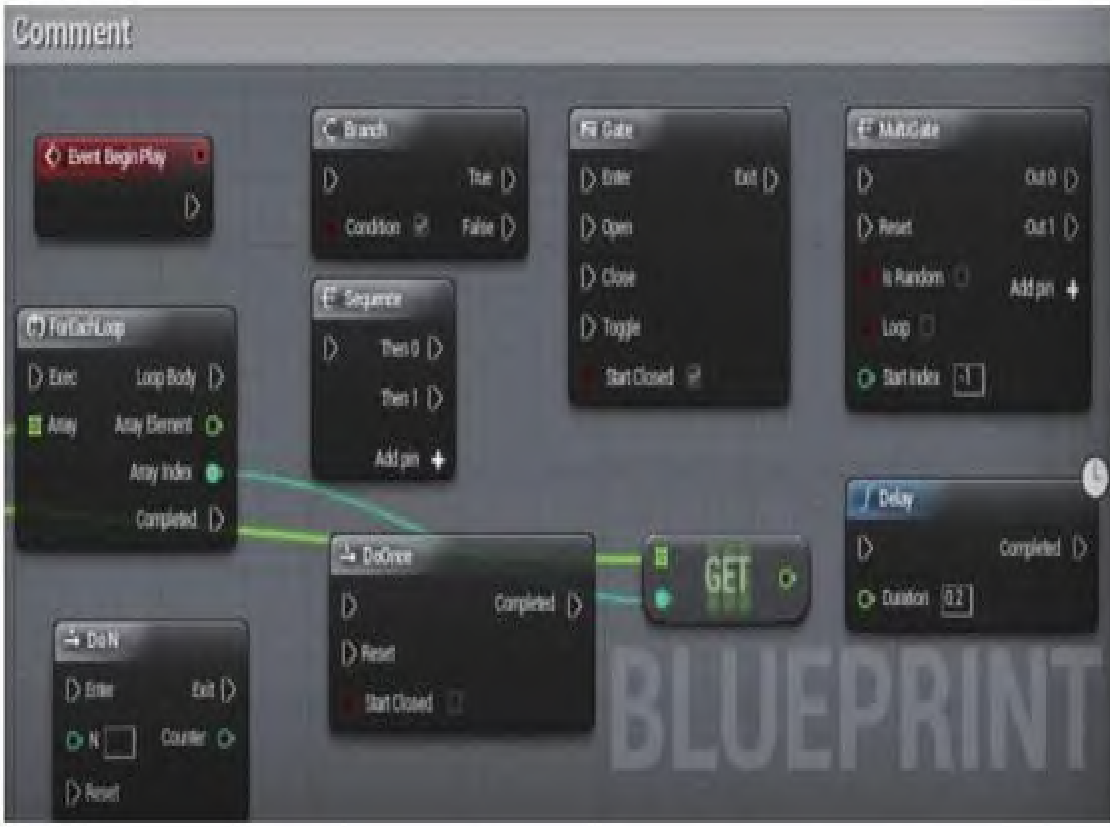



图3-60

| 操　作 | 命　令 |
| --- | --- |
| Array Get Node（获得数组节点） | A+左击 |
| Branch Node（分支节点） | B+左击 |

续表

| 操 作 | 命 令 |
| --- | --- |
| Comment Box Node（注释框节点） | C |
| Delay Node（延迟节点） | D+左击 |
| Sequence Node（序列节点） | S+左击 |
| Gate Node（门节点） | G+左击 |
| For-Each Loop Node(For-Each 循环节点） | F+左击 |
| Multi-gateNode（多门节点） | M+左击 |
| Do N Times Node（执行N次节点） | N+左击 |
| Do Once Node（执行一次节点） | O+左击 |
| BeginPlay Event（开始运行事件） | P+左击 |

## 3.2.2 蓝图搜索

### 1.搜索蓝图

在蓝图编辑器中时，可单击工具栏中的Search或使用Ctrl+F呼出搜索结果（Find Results）窗口。可在此处搜索蓝图中匹配搜寻条件的节点、引脚、引脚值、图表、变量和变量值，也可搜索动画图表。

右击蓝图中的节点或My Blueprint窗口中的元素，然后单击Find References同样可以打开Find Results窗口，在搜索栏填入元素名称和元素的MemberGuid。即使项目其他部分拥有多个该名称，这也能确保你搜索的是特定变量或函数。

搜索为不同步过程，意味着搜索完成后即可使用编辑器。也可同时在不同蓝图中进行多项搜索。它利用搜索数据的资源注册表，因此最近编入索引的数据会固定地和资源文件结合。

此工具默认搜索当前蓝图，取消勾选Find in Current Blueprint Only复选框后即可搜索项目中的所有蓝图。蓝图未在系统中编入索引时将显示下面的提示，如图3-61所示。

图3-61

单击Index All按钮后进程将变得极慢，使编辑器停滞。因为此项将加载所有未编入索引的蓝图，并将它们重新保存以缓存搜索数据。如不希望重新保存内容，或内容受源控制保护无法移出，它将直接加载所有蓝图资源，缓存内存中最新的搜索数据。如未重新保存内容，每次打开编辑器时都需要执行Index All，如图3-62所示。

### 2. 高级搜索语法

过滤器是一项高级搜索功能，可针对蓝图中特定的数据子集进行搜索，例如，可将拥有特定命名节点的蓝图或带特定名称和标记集的属性单独隔离出来。它们可被嵌套和组合，形成每个特定需求的高级查询。部分过滤器只能在其他过滤器中使用，但所有过滤器均可用作起始点。下表是搜索标记的非详尽列示，以及用作的数据类型，无须使用过滤器便可对所有标记进行搜索。

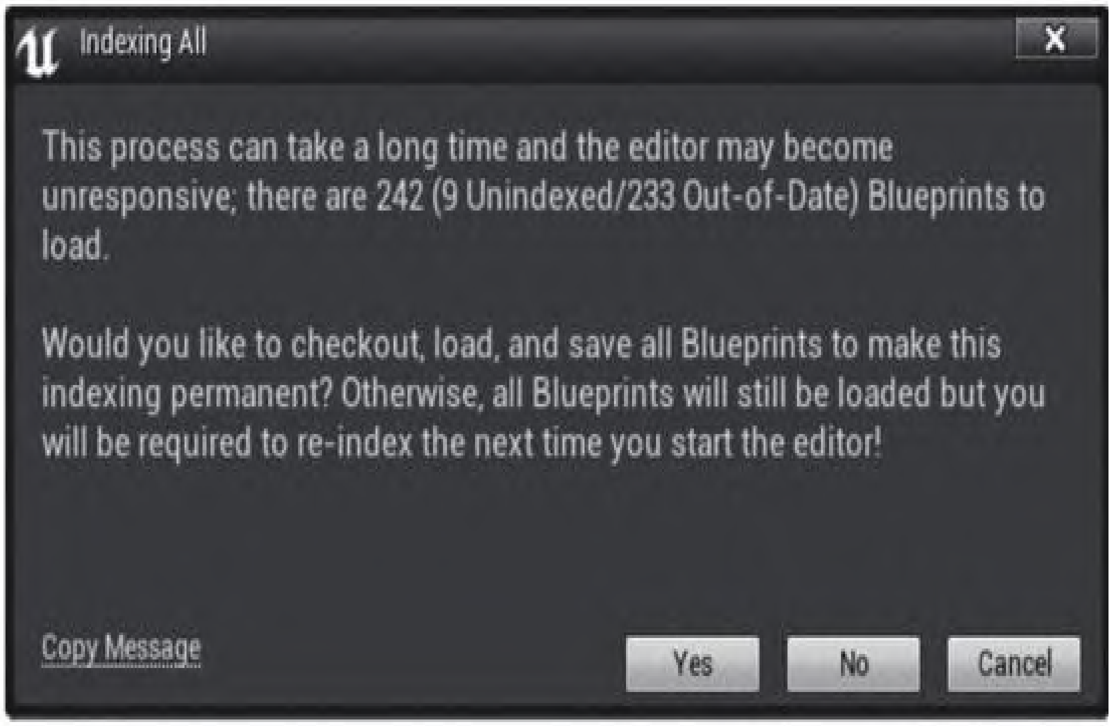



图3-62

某些项目只能通过标记搜索，防止不使用标记进行搜索时出现混乱。这些项目以下标记为“（Explicit）”。当前仅可对用户添加的成员变量进行过滤。

以搜索字符串Nodes（Name=Coin）为例，它将找到蓝图中所有命名包含“coin”的节点。

| 过滤器 | 搜索标记 | 子过滤器 |
|---|---|---|
| 蓝图 | Name<br>ParentClass (Explicit)<br>Path (Explicit)<br>Interfaces (Explicit) | Graphs<br>Functions<br>Macros<br>EventGraphs<br>Nodes<br>Pins<br>Variables/Properties<br>Components |
| Graphs<br>Functions<br>Macros<br>EventGraphs | Name<br>Description | Nodes<br>Pins<br>Variables/Properties（针对本地变量） |
| 节点 | Name<br>ClassName (Explicit)<br>Comment | 引脚 |
| Pins<br>Variables/Properties<br>Components | Name<br>DefaultValue<br>IsArray (Explicit)<br>IsReference (Explicit)<br>IsSCSComponent (Explicit)<br>PinCategory (Explicit)<br>ObjectClass (Explicit) | |

PinCategory是引脚的类型，诸如“布尔”“字符串”“Actor”“MyBlueprint”。Object Class是引脚/变量/组件所代表的结构体或对象。

All子过滤器：

All是一个特殊的子过滤器。嵌套在另一个过滤器中时，它可使其中的数据对通过过滤器的对象的所有子类进行测试。

范例：Graphs（Name=MyFunction && All（Return））

包含名称“MyFunction”的所有图表将对其所有子类（节点、本地变量和引脚）测试字符串“Return”。

## 3.2.3　蓝图调试1

蓝图调试是一种非常强大的功能，允许你在Play In Editor（在编辑器中运行）或Simulate In Editor（在编辑器中模拟）模式下使用断点来中止游戏执行过程，并单步调试任何蓝图图表或关卡蓝图。

### 3.2.3.1　调试控件

调试控件允许你在Play In Editor和Simulate In Editor模式下控制游戏的执行过程。当游戏运行时，会在蓝图编辑器中启用这些控件。根据所调试的蓝图及当前调试状态的不同，会在工具条上出现不同的调试按钮，如图3-63所示。

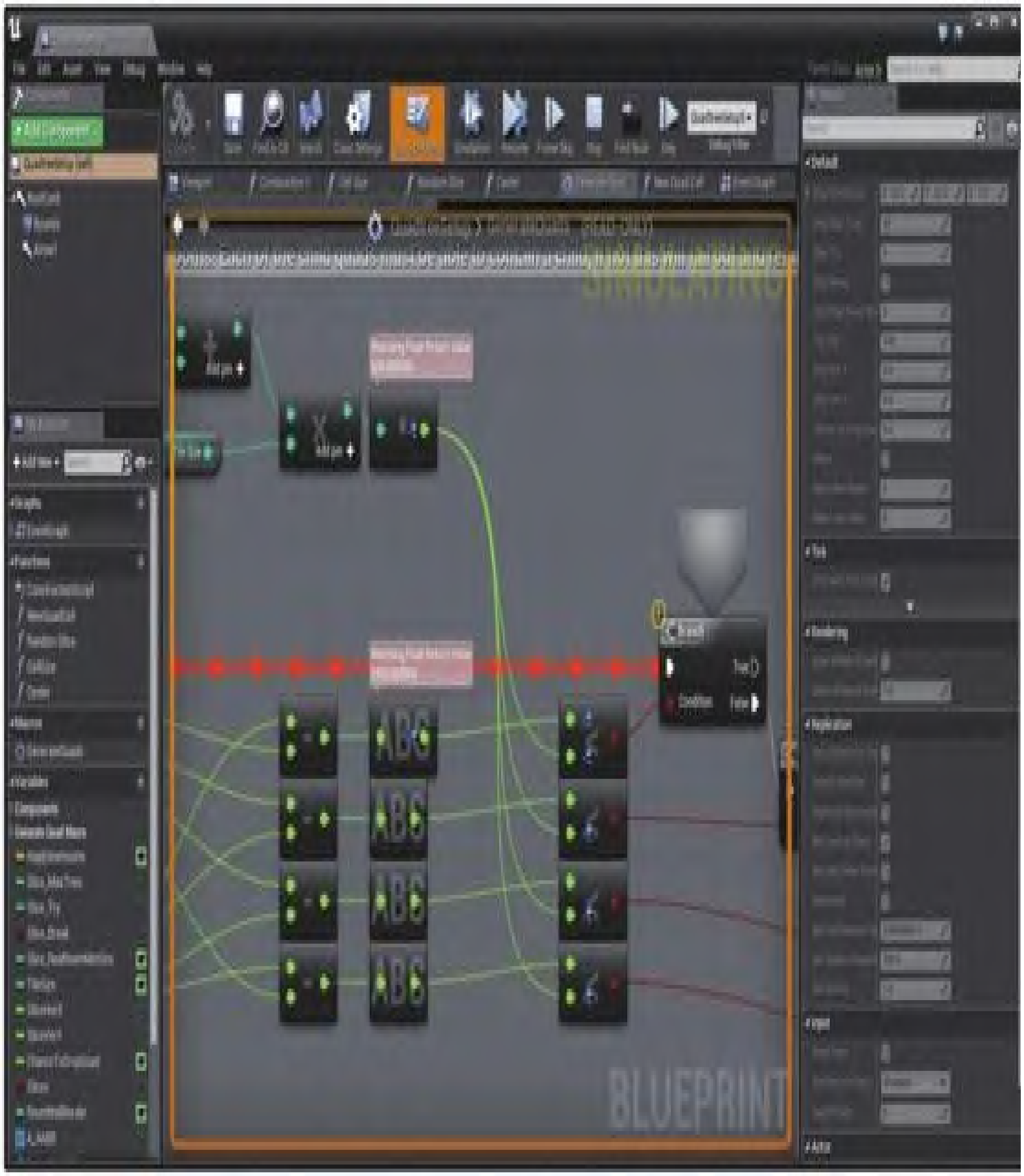


图3-63

当激活Simulate in Editor或Play in Editor模式时，Debug（调试）选卡和Blueprint Debugger（蓝图调试器）都将显示情境关联的调试按钮。其中Debug（调试）选卡可以从蓝图编辑器中的Graph（图表）模式下打开。

调试控件允许你在Play In Editor和Simulate In Editor模式下控制游戏的执行过程。当游戏运行时会启用这些控件。根据所调试的蓝图及当前调试状态的不同，会在工具条上出现不同的调试按钮。一些控件仅在遇到相关断点时才会被激活，如图3-64所示。

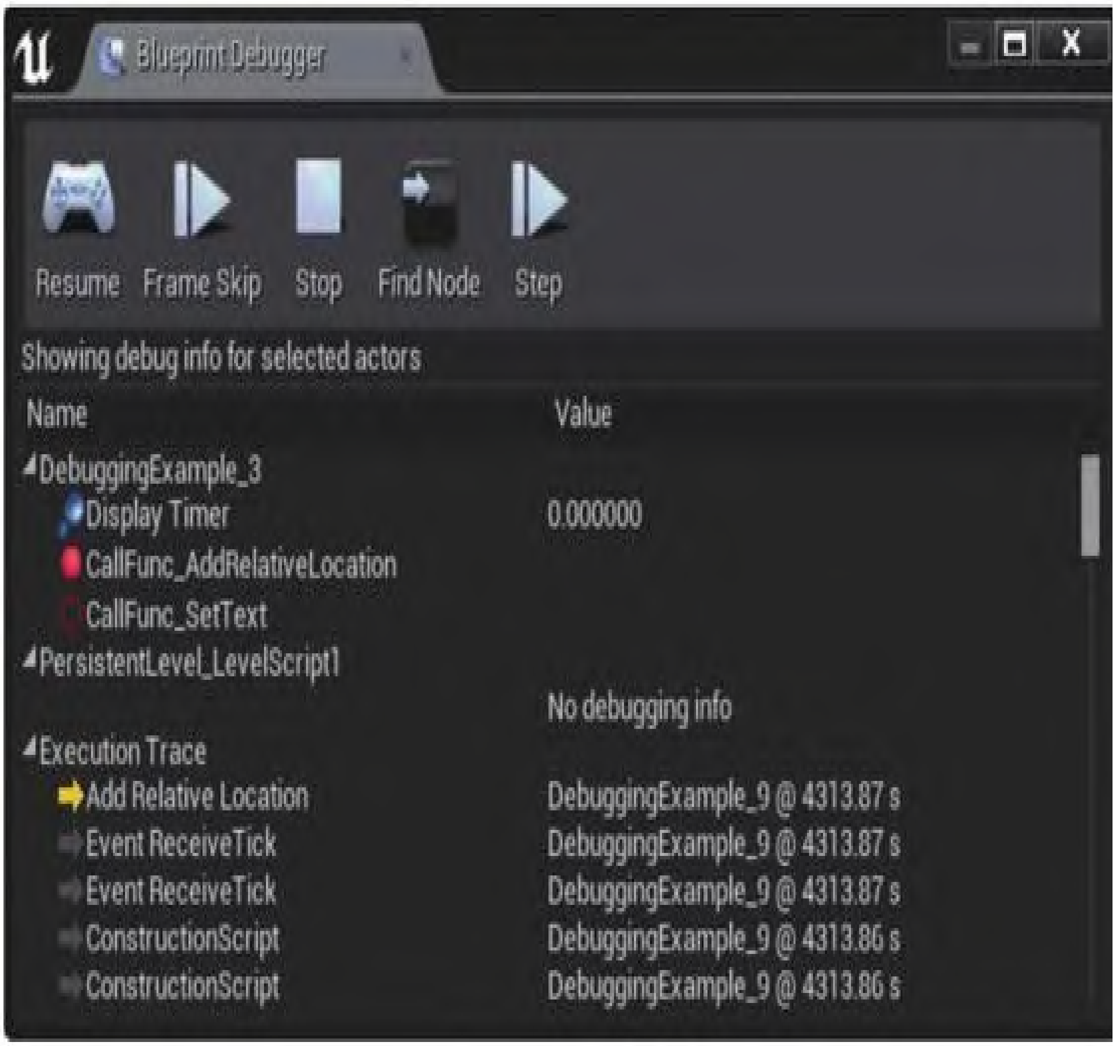



图3-64

### 3.2.3.2 断点

当从编辑器中运行游戏时，Breakpoints（断点）可以在任何点处中止游戏的执行过程。属于一个蓝图或关卡蓝图的所有图表的所有断点都会显示在Debug（调试）选卡中。该列表中的断点名称都有超链接，你可以单击它们，并直接跳转到Graph（图表）选卡中和那个断点相关的节点处。

你可以在要执行的图表中任何节点上设置断点，比如Function Calls（函数调用）和Flow Control（流程控制）操作，只需通过右击节点并选择Add breakpoint（添加断点）即可设置断点。当已经在一个节点上设置了断点时，会在该节点的左上角上显示一个图形，如图3-65所示。

通过右击断点并选择Disable breakpoint（禁用断点）或者单击Debug（调试）菜单中断点旁边的图标将其关闭，可以禁用断点。这个处理也

可以通过右击Debug（调试）选卡中的断点并选择Disable breakpoint（禁用断点）来完成。

这样做时断点会留在原处，但是当执行该节点时它不会使游戏停止。已禁用的断点仅显示为一个断点轮廓，如图3-66所示。

图3-65

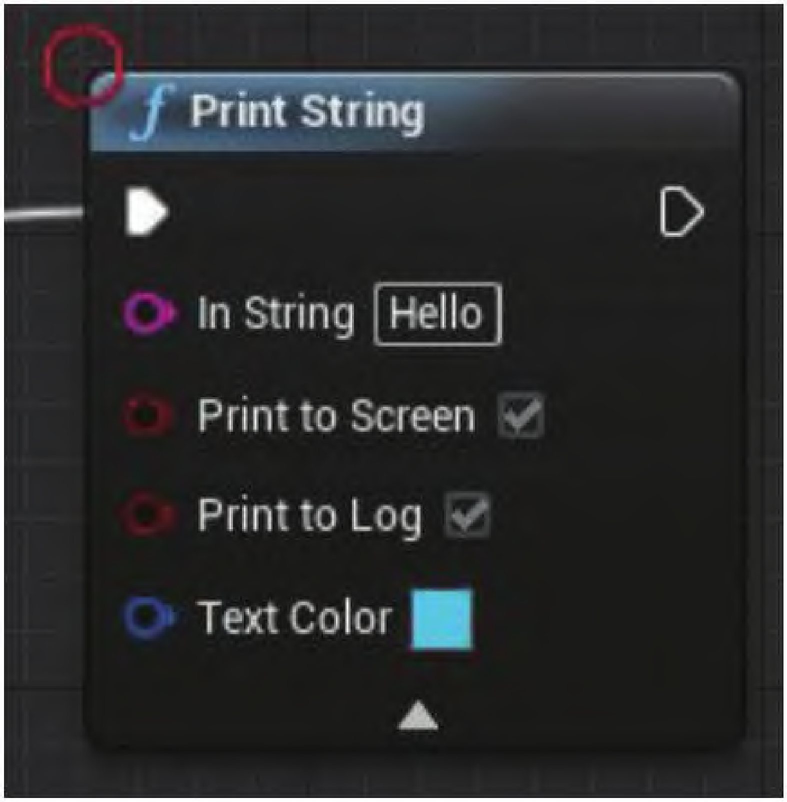



图3-66

要想启用已禁用的断点，请右击该节点并选择Enable breakpoint（启用断点）或单击Debug（调试）选卡中断点附近的图标。这个处理也可以通过右击Debug（调试）选卡中的断点并选择Disable breakpoint（禁用断点）来完成，如图3-67所示。

图3-67

通过右击一个节点或者右击Debug（调试）选卡中的断点，并选择Remove breakpoint（删除断点），可以删除断点。

当碰到一个断点时，图表将专注于这个包含该断点的节点，并在断点上显示一个指示符，如图3-68所示。

另外，视口将显示一个覆盖层，来告知玩家游戏已中止，正在进行调试，如图3-69所示。

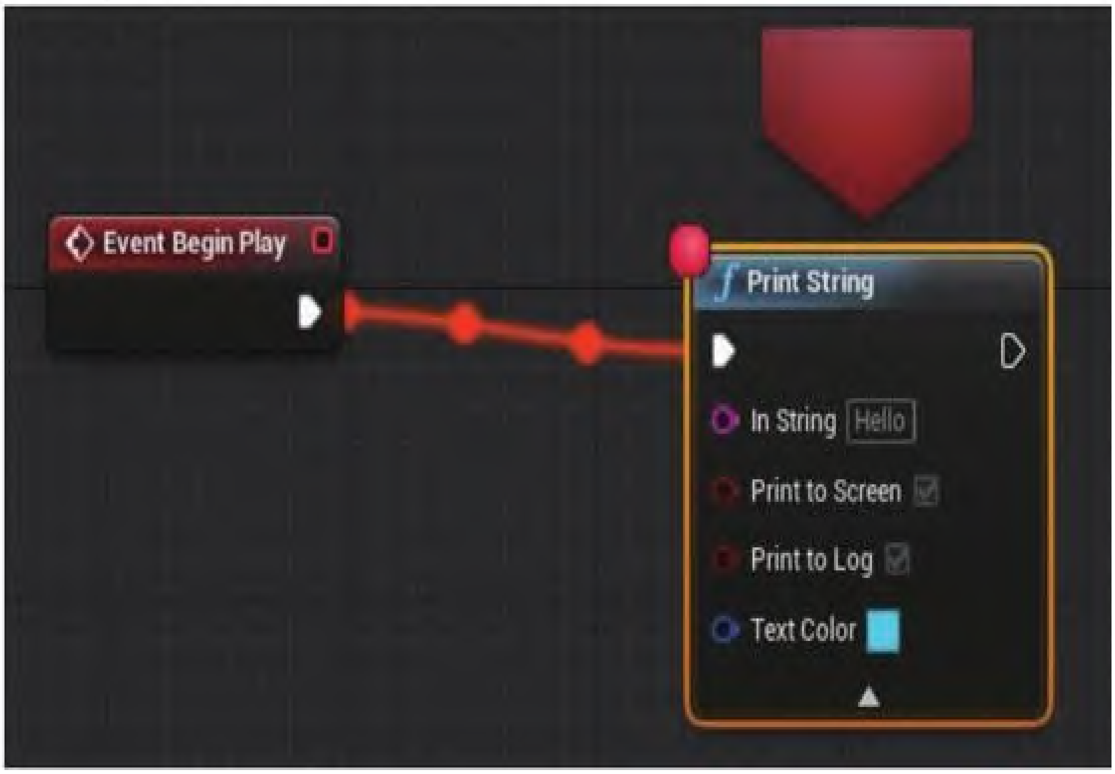



图3-68

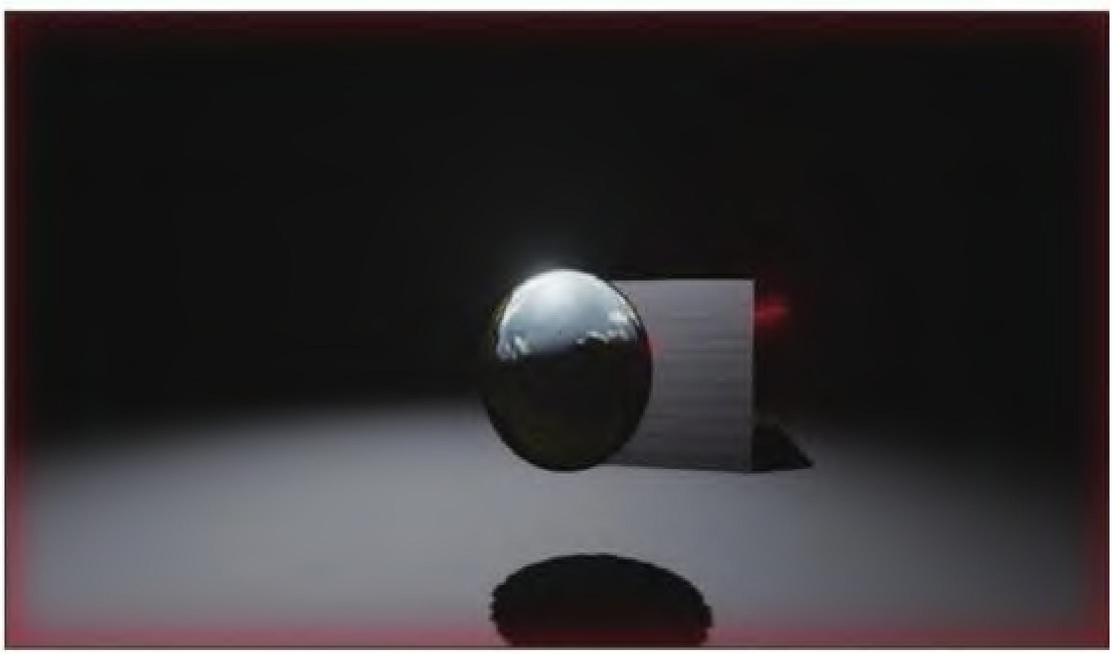


图3-69

#### 3.2.3.3 调试选卡和蓝图调试器

Debug（调试）选卡显示了设计师指定的重要信息，信息以断点、查看值及属于已经执行过的蓝图的所有节点的堆栈追踪的形式显示。当使用断点时，这个窗口也显示了用于控制游戏执行的控件。

### 1.变量值查看器

Watch values（查看值）允许你在执行过程中快速并轻松地查看变量的值。当调试遇到一个断点后，所有选中的Actor及关卡蓝图的查看值都会显示在调试窗口中，并且随着你单步调试整个图表，这些查看值会随之更新，如图3-70所示。

在图表中，正在查看的值可以直接以气泡的形式显示在变量的上方，如图3-71所示。

要想启用显示查看信息气泡的功能，仅需简单地单击变量名称旁边的放大镜（）即可。

### 2.执行追踪

Execution Trace（执行追踪）栈显示了所执行的节点的列表，最近执行的节点显示在顶部，如图3-72所示。

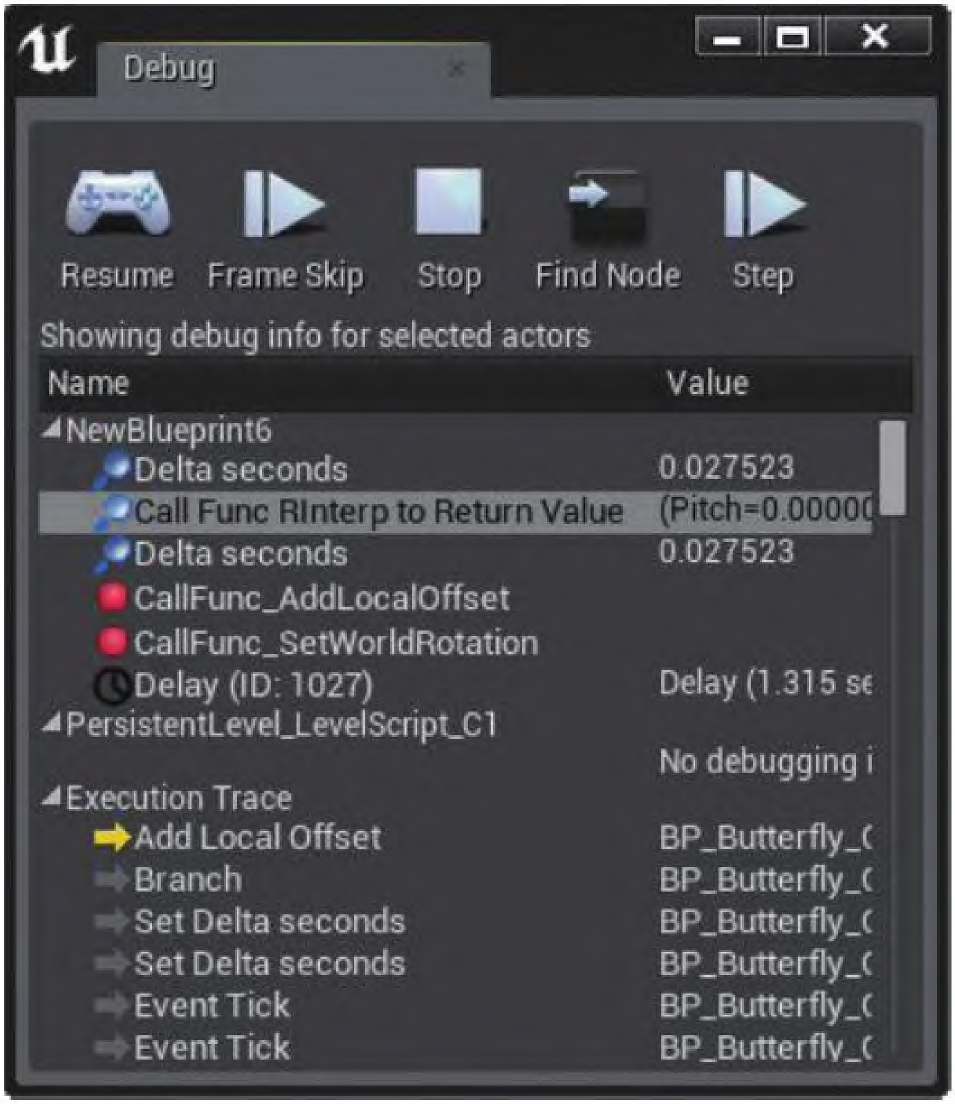

Debug
Resume
Frame Skip
Stop
Find Node
Step
Showing debug info for selected actors
Name
Value
NewBlueprint6
Delta seconds
0.027523
Call Func RInterp to Return Value
(Pitch=0.00000
Delta seconds
0.027523
CallFunc_AddLocalOffset
CallFunc_SetWorldRotation
Delay (ID: 1027)
Delay (1.315 se
PersistentLevel_LevelScript_C1
No debugging i
Execution Trace
Add Local Offset
BP_Butterfly_(
Branch
BP_Butterfly_(
Set Delta seconds
BP_Butterfly_(
Set Delta seconds
BP_Butterfly_(
Event Tick
BP_Butterfly_(
Event Tick
BP_Butterfly_(

图3-70

Watching float Delta Seconds
0.036611
Delta Seconds

图3-71

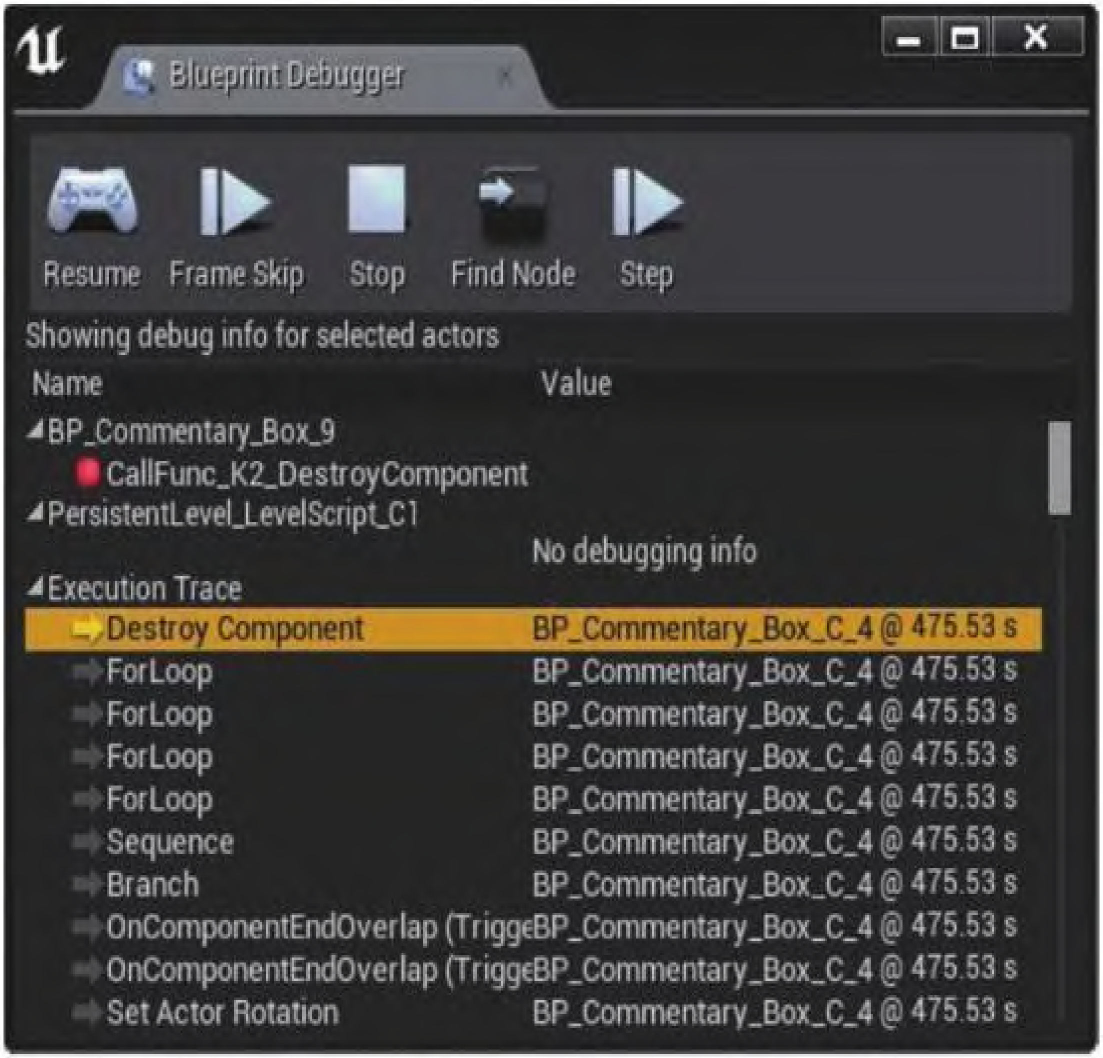



图3-72

这个列表会随着你单步调试整个图表而进行更新。

## 3.2.4　蓝图调试2

在使用蓝图时，可能出现预料之外的状况。为诊断这些问题，蓝图可视化脚本工具除自带调试器外，还拥有最强大的功能之一：在图表中为节点添加断点（Breakpoints）。在图表中为节点添加断点后，可在编辑器中正常进行（或模拟）游戏。到达存在断点的节点后，游戏将暂停并跳至图表中的节点，以便对脚本进行检查，确认问题出于何处。

另一个实用调试功能是Watch Values，它可将蓝图中的一个（或多个）变量标记到观察（Watch），以便在游戏运行时在蓝图中看到变量数值。这对

调试变量错误计算或数值十分重要，因为可以在变量实时变化时发现错误的源头。

在本节中，你将学习如何为蓝图设置调试，以及用于诊断问题的其他方法。

#### 3.2.4.1　启用调试

如需启用蓝图调试，首先必须确定需要进行调试的蓝图实例位于关卡中。

在蓝图中，单击Debug Object下拉菜单，然后选择调试器中的实例，如图3-73所示。

图3-73

选定对象后，单击放大镜图标可在关卡中跳至该对象，如图3-74所示。

图3-74

在编辑器中进行游戏时，将蓝图放置于另一个窗口中，则可看到脚本执行时跳动的“活性线条”。

在上例的关卡中有两个火焰效果蓝图，各自拥有不同的停止时间。在Event Begin Play节点上（游戏启动后），Delay节点用于在变量指定的时间结束后停止火焰和相关音效。

游戏启动后（当跳动线条离开Event Begin Play节点，进入Delay节点时）可在左边窗口中看到脚本的执行。在指定时间后，跳动线条进入第三个节点，而该节点用于停止其中一个火焰效果。

### 3.2.4.2　断点

在任意执行节点上右击选择Add Breakpoint，可添加一个断点至蓝图脚本，如图3-75所示。

图3-75

除右击的方法外，还可在选定的节点上按F9键切换断点的开关。

断点被添加至节点后，节点的左上角将出现一个红圈，如图3-76所示。

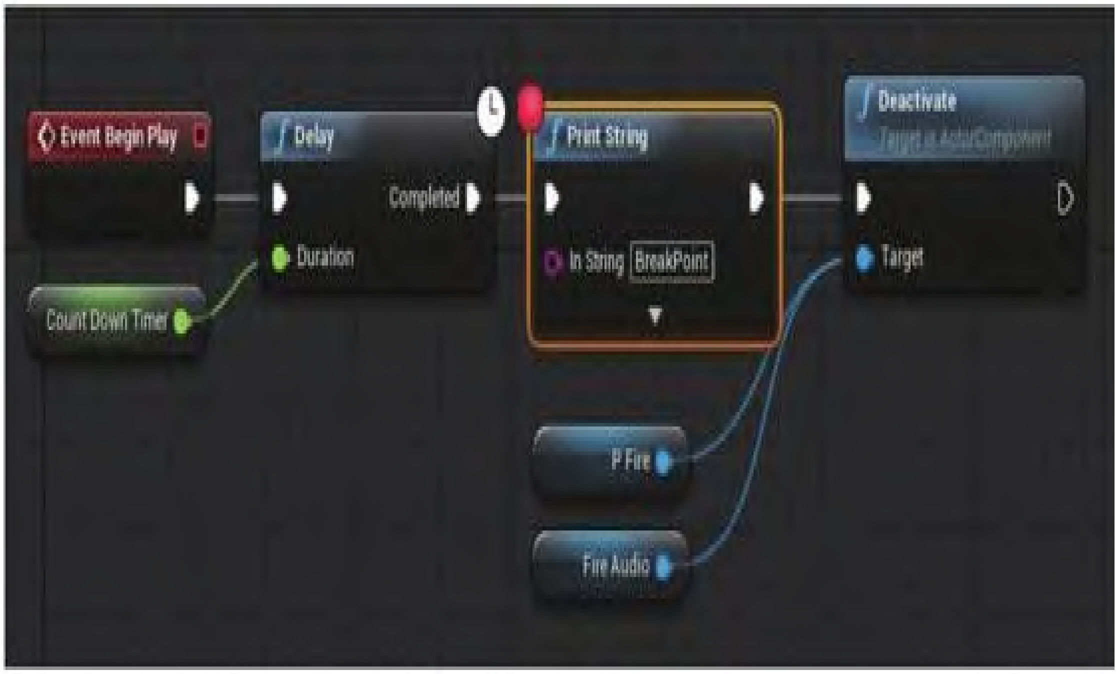



图3-76

游戏进程中脚本到达此点时，游戏将暂停并聚焦于此节点，如图3-77所示。

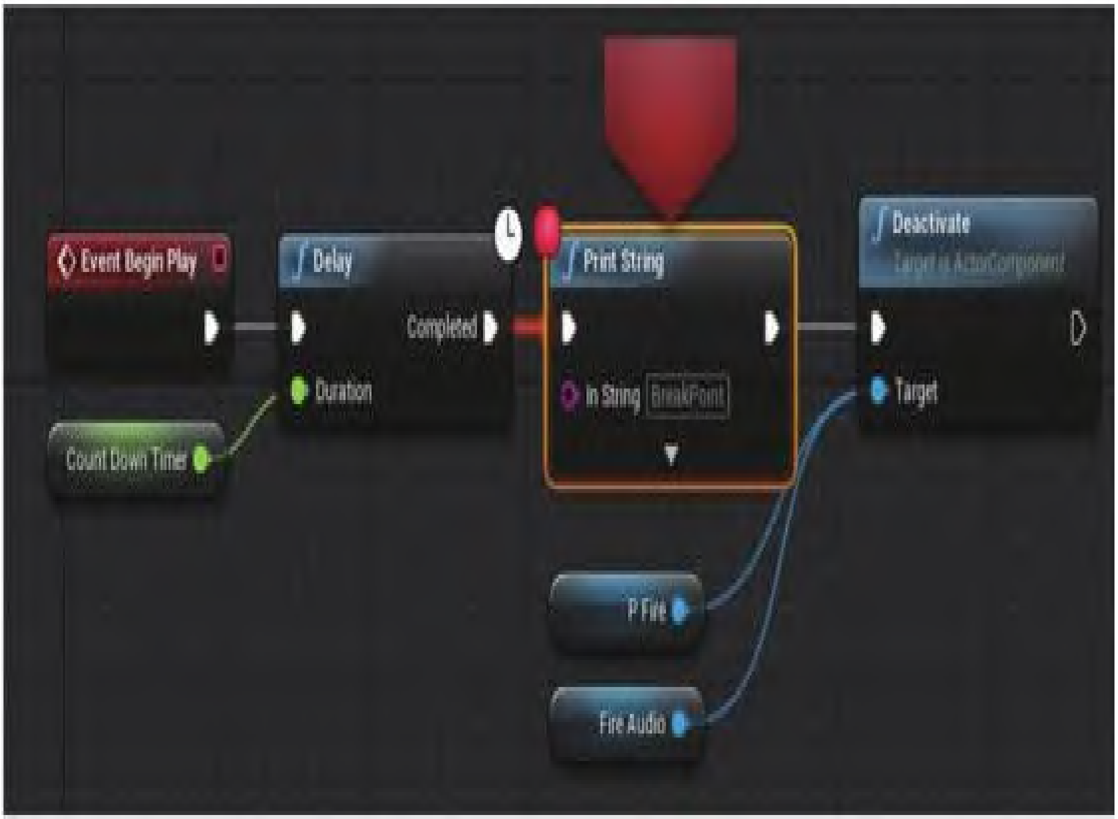



图3-77

也可临时禁用断点，记录之前哪些节点指派了断点，如图3-78所示。

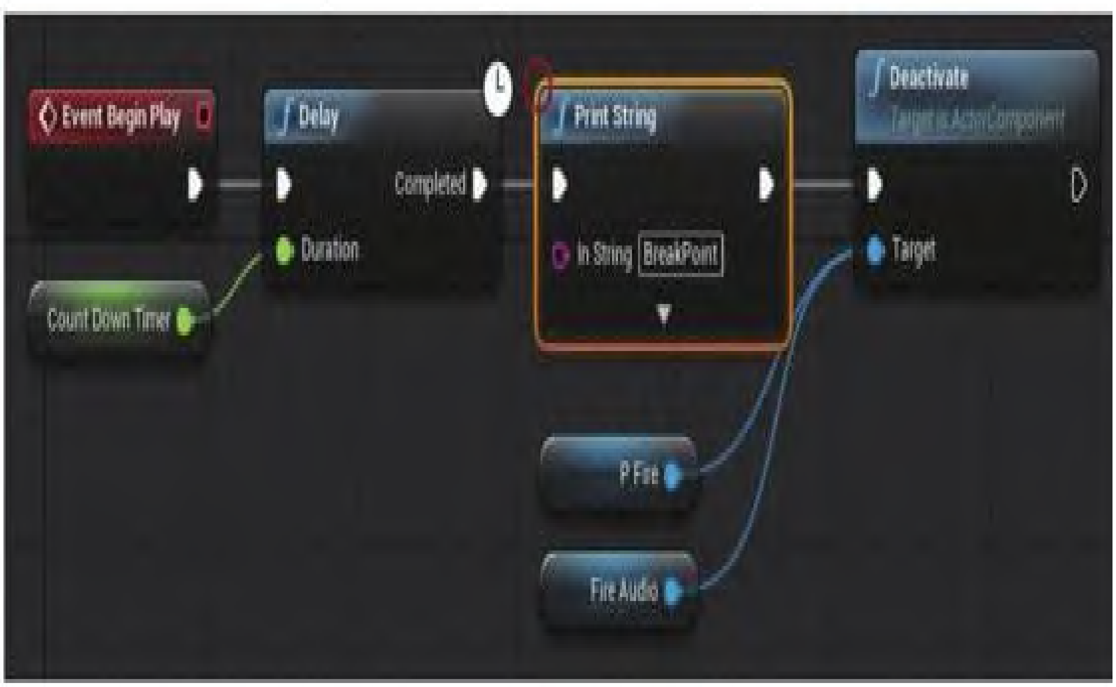


图3-78

断点被禁用后，到达节点时将忽略断开指令。

Debug菜单中的选项可用于禁用/启用或删除全部断点，如图3-79所示。

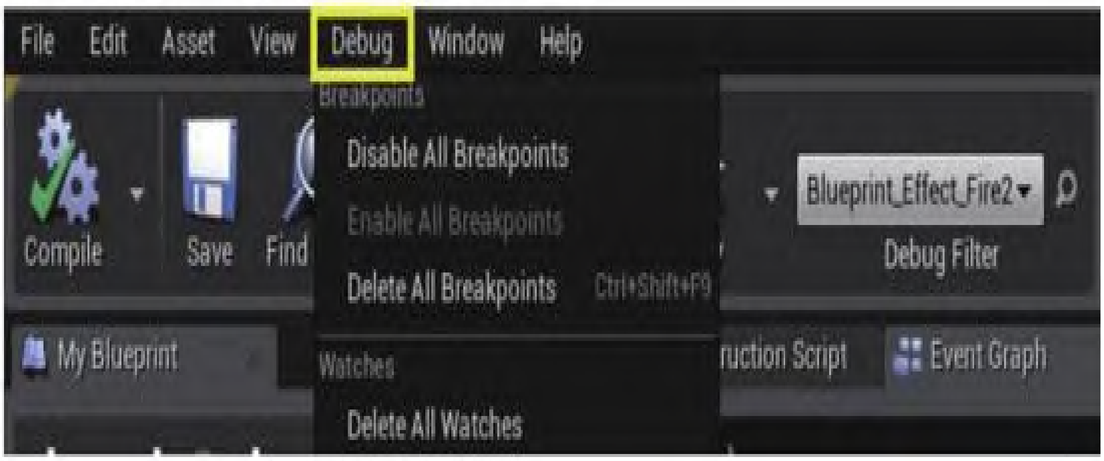



图3-79

### 3.2.4.3　调试控制

当蓝图到达断点时，可选额外控制选项，便于进行蓝图脚本导航，如图3-80所示。

图3-80

顶部从左至右，Resume用于在编辑器中继续游戏和退出调试模式。Frame Skip用于以单帧前进游戏进程；Stop用于编辑器会话中完全停止游戏。Find Node选项用于从游戏会话跳至当前活动的节点；Step用于在当前执行流中前进至下个节点。

在节点上放置一个断点后即开始脚本进程，而在到达当前执行流的终点时，将自动返回游戏。

调试的常规工作流是为节点添加一个断点，然后使用Step在脚本中前进，找到问题源头。放置断点需要注意的是在脚本中无法后退，须将其放置在问题发生之前，才能完整地看到问题的前后全貌。

### 3.2.4.4　观察数值

在变量上右击并选择Watch this value选项即可观察变量值，如图3-81所示。

选择观察变量后，变量访问之处将出现放大镜图标，如图3-82所示。

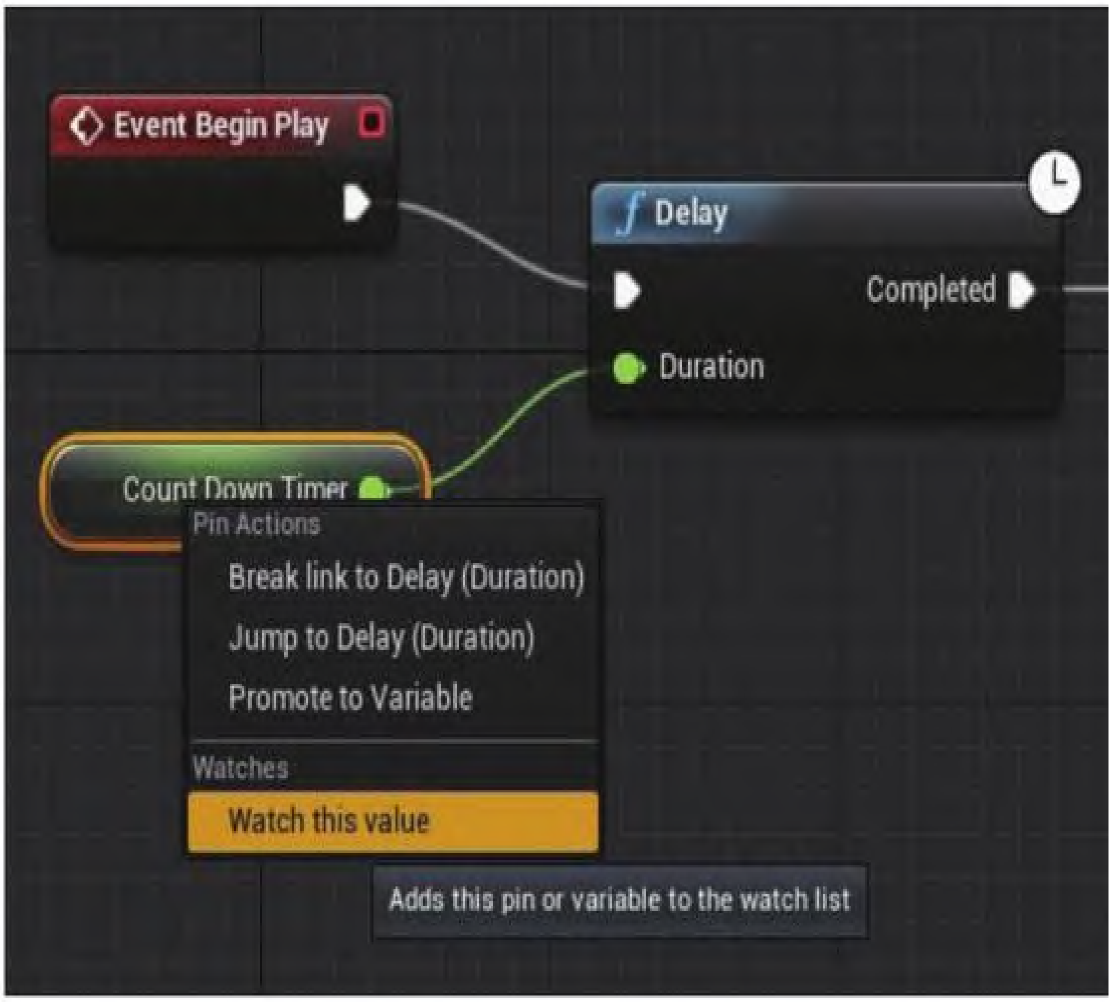



图3-81

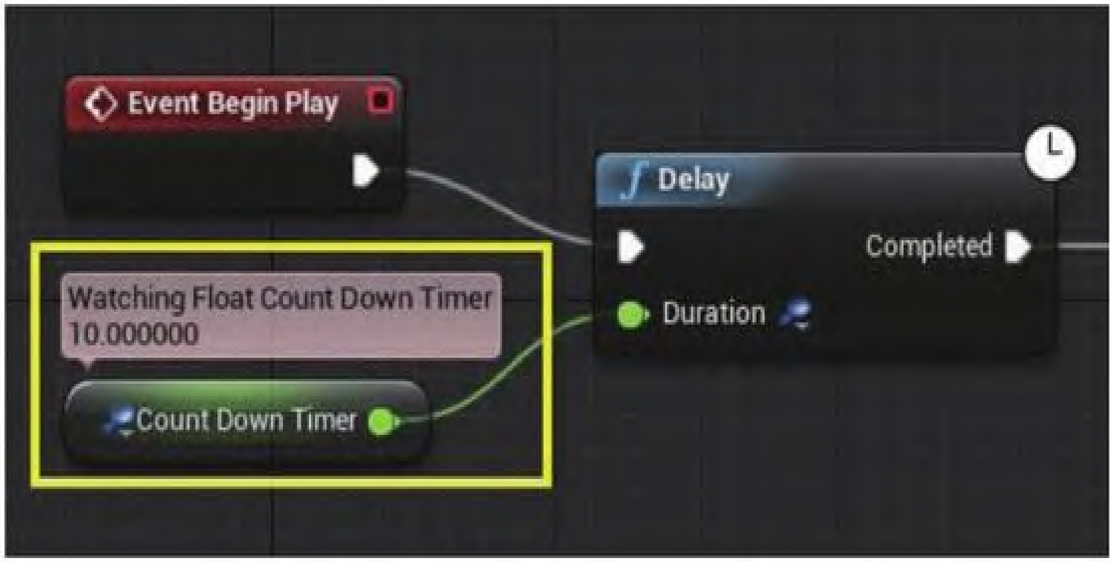



图3-82

变量上方的浮动文本框显示变量的当前数值。在编辑器中进行游戏时，如变量发生变化，它也将反映在文本框中，以便查看其变化和当前数值。观察变量的选项不限于数值式变量，任意变量均可被观察，以便了解数值（下图中的Boolean正被观察，我们可以了解其当前状态），如图3-83所示。

图3-83

可通过Debug菜单中的Delete All Watches选项从所有变量上移除观察符号，如图3-84所示。

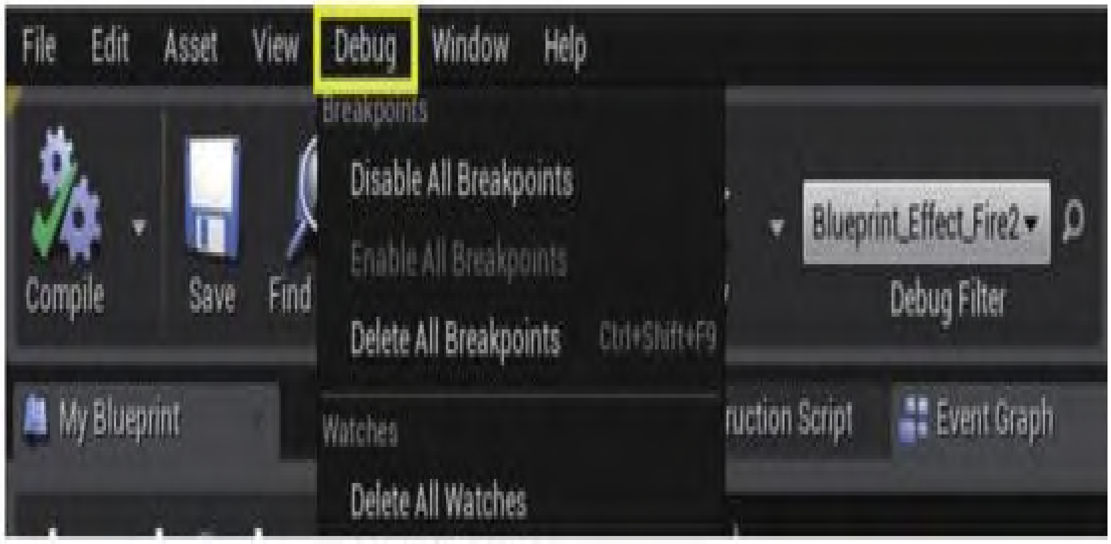



图3-84

### 3.2.4.5　蓝图调试器标签

也可通过Blueprint Debugger对蓝图进行调试，该选项位于Window/Developer Tools部分，如图3-85所示。

不在编辑器中进行游戏时，Blueprint Debugger可显示所有被观察的变量或指派的断点（下图黄色框中的内容），如图3-86所示。

按住Shift键单击关卡中的Actors可将多个蓝图添加至Blueprint Debugger标签。

在编辑器中进行（或模拟）游戏时，调试器的下部将被填入内容。这部分为执行追踪（Execution Trace），如图3-87所示。

Window
Help
Graph Editor
Toolbar
Details
Debug
Palette
My Blueprint
Compiler Results
Find Results
Viewport
Blueprint Defaults
General
Content Browser
Developer Tools
Plugins
Layout
Reset Layout...
Save Layout
Enable Fullscreen
Shift+F11
Blueprint_Effect_Fire
Debug Filter
Event Graph
Debug
Blueprint Debugger
Collision Analyzer
Debug Tools
Log
Message Log
Output Log
Miscellaneous
Class Viewer
Device Manager
Device Profiles
Session Frontend
Widget Reflector
Open the Blueprint Debugger tab.

图3-85

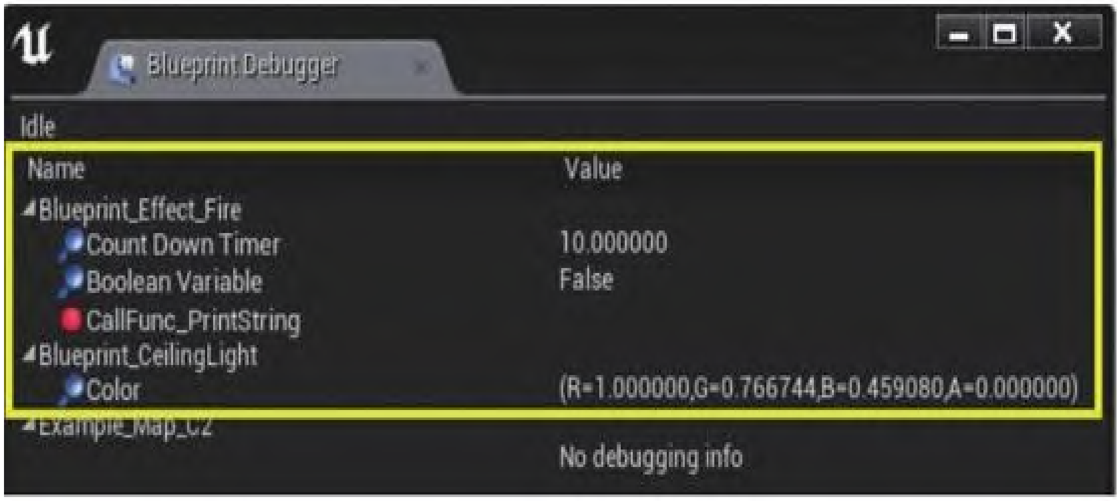



图3-86

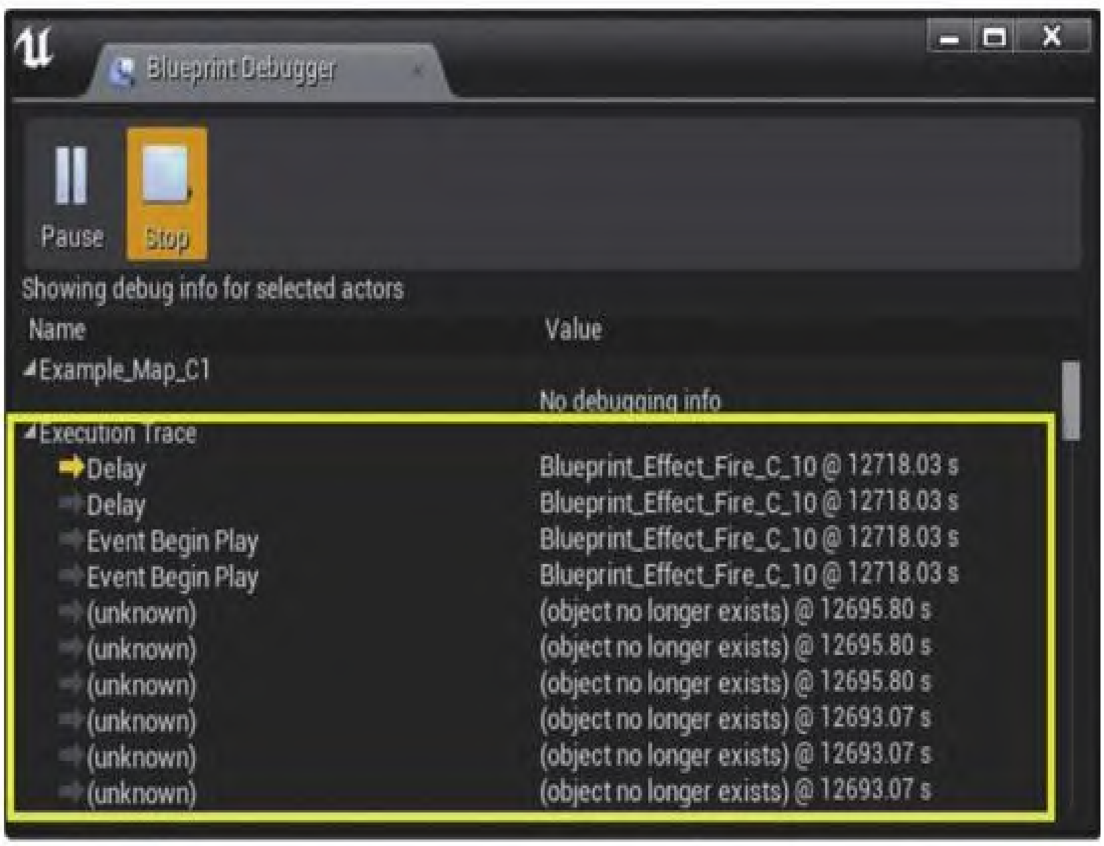



图3-87

它是所有被执行指令的日志记录，以生成时间排序（上方为最近生成）。

到达断点时，可在蓝图调试器标签中直接访问本节中提及的调试控制。

### 3.2.4.6 编译错误

编译蓝图时出现编译错误（下图黄色框所示）存在多种原因，如图3-88所示。

图3-88

出现这种情况后，进行游戏时将出现类似于下面的提示信息，如图3-89所示。

图3-89

存在错误的蓝图将同样被反映在关卡编辑器视口中，如图3-90所示。

图3-90

在这种情况下，可能无法通过调试方法成功确定错误。在编译报错的蓝图中，引起错误的节点将被标记，以便确定问题所在之处，如图3-91所示。

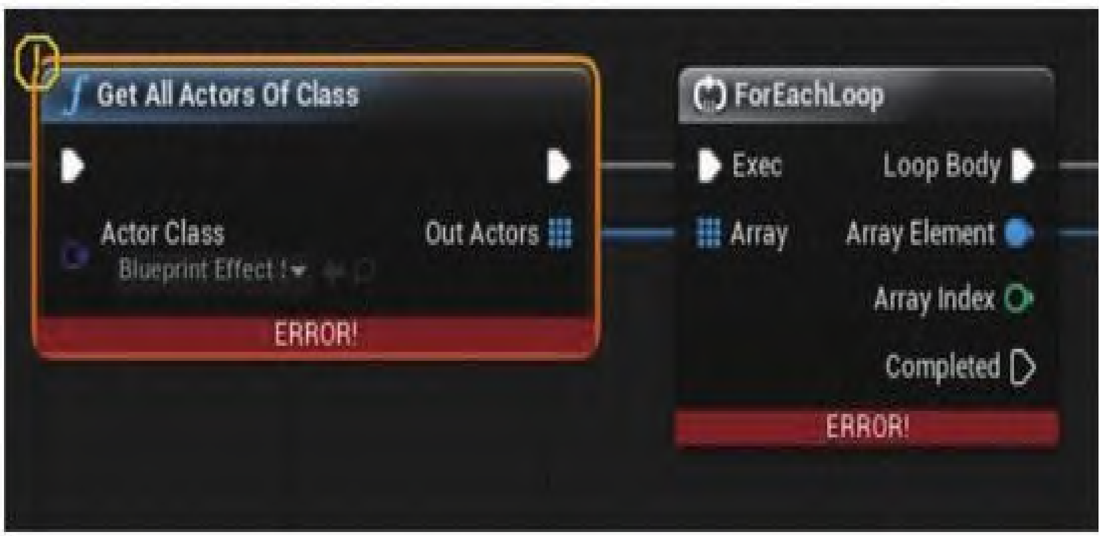



图3-91

可在Compiler Results窗口中查看关于报错的更多信息，如图3-92所示。

图3-92

出现错误时Compiler Results窗口将自动弹出。可单击放大镜跳至错误所在位置和问题节点。为帮助解决问题，将显示错误的文本描述。

## 3.2.5　注释

注释可被直接添加到单个蓝图节点上，也可以注释框的形式添加到组相关节点上并提供其功能的描述信息。它们可以单独以其管理用途来让图表更易读懂，但它们也可以用来提供信息，正如添加注释到节点一样，使用注释或注释框可以把文本描述内容添加到节点上。

### 3.2.5.1　节点注释

**1.创建节点注释**

如需直接在蓝图图表中添加注释：

Step 1　在节点上右击。

Step 2　在出现的菜单中，在Node Comment（节点注释）文本框位置处输入注释，然后按下回车，如图3-93所示。

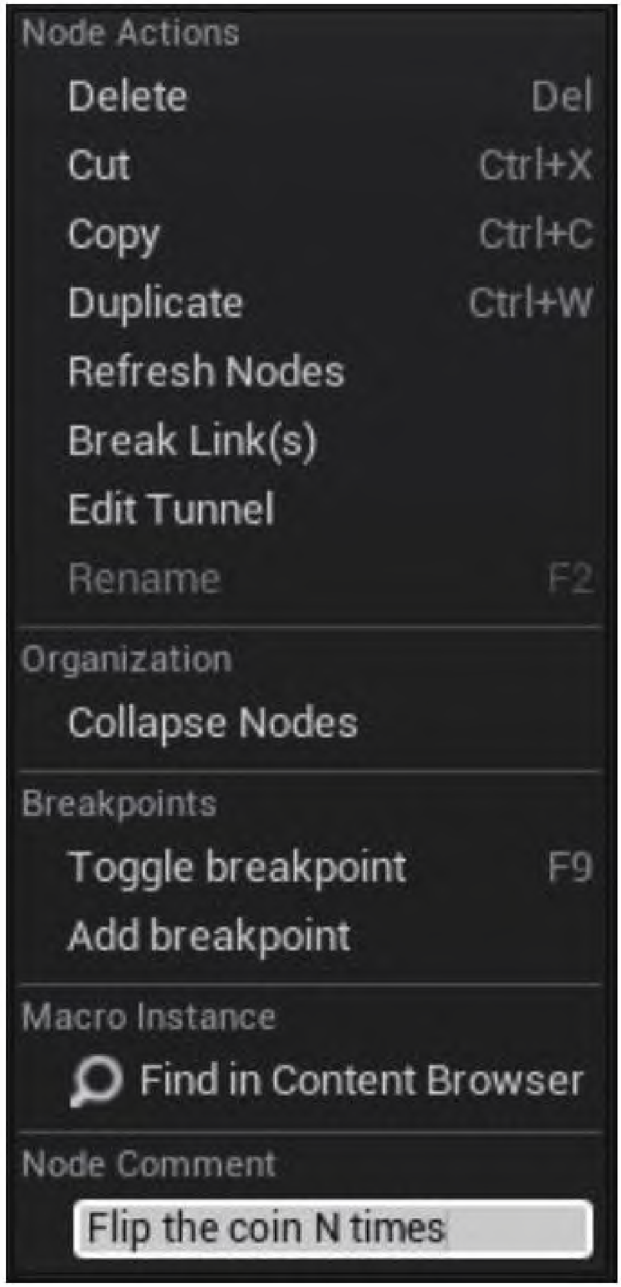



图3-93

你的注释会出现于节点上方的小注释气泡中。如对图表进行缩放，节点注释将不会相应缩放。

**2. 编辑并移除节点注释**

可以再次右击节点，在右击出现的菜单中变更Node Comment（节点注释）文本框的内容来改变注释。删除文本并按下回车将移除注释。

### 3.2.5.2　注释框

1.创建注释框

**Step 1**　选择一个你想要进行注释的节点组。你也可以先创建一个空白的注释框，并在稍后对节点附近的注释框进行排列，或者将其作为悬浮注释放置于指定区域。

**Step 2**　右击并选择Create Comment from Selection（从选择项中创建注释）（如果未选中任何节点则选择Add Comment（添加注释））。此时将出现新的注释框，如图3-94所示。

也可以单击“C”来创建注释。

**Step 3**　将你的注释输入出现的注释框的页眉位置，如图3-95所示。

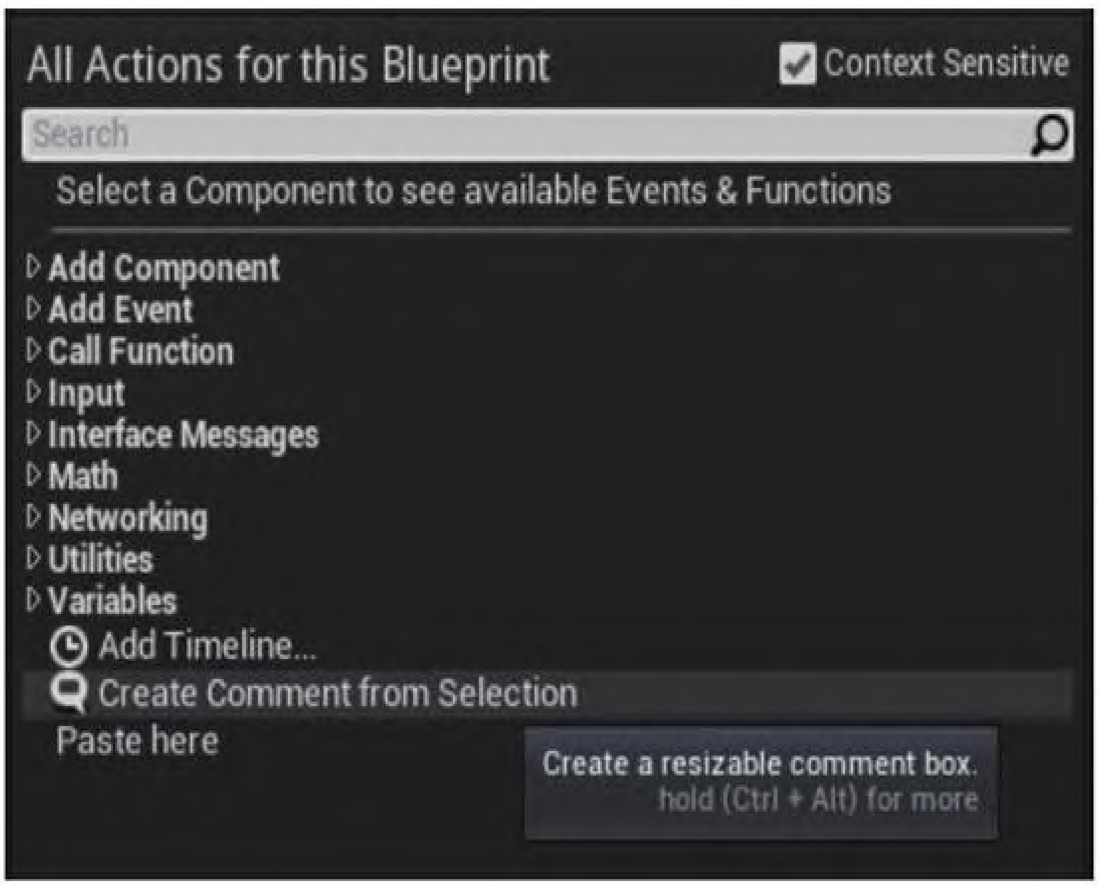



图3-94

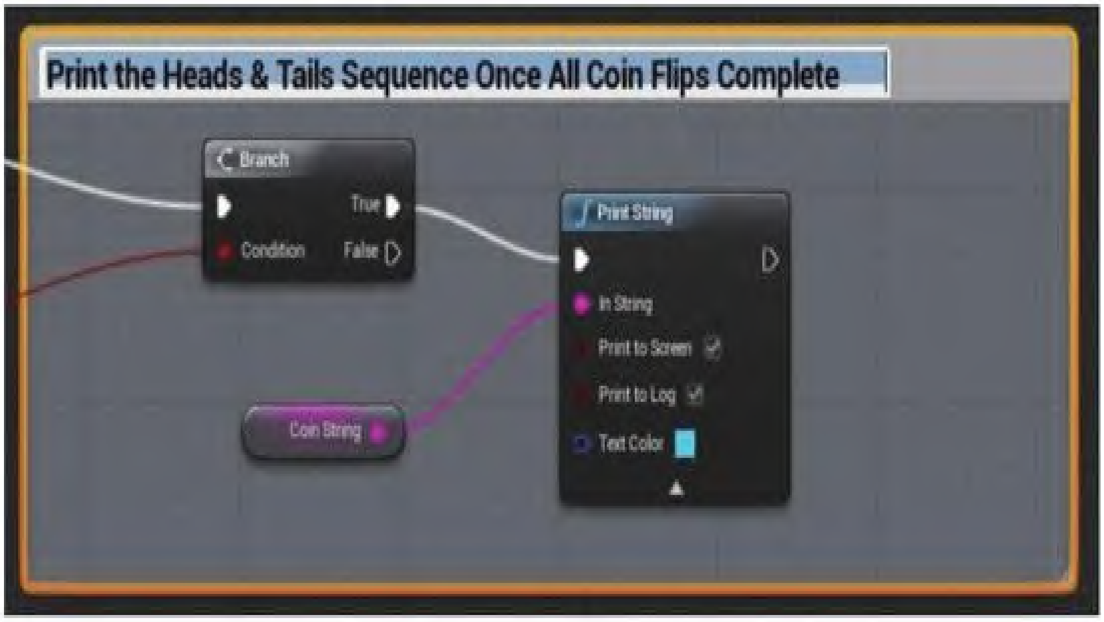



图3-95

**Step 4**　你可以通过拖动注释框的右下角来重新定义注释框尺寸。请注意如果你尝试移动注释框，则任何位于框体内的节点将会一起移动。这对于重新排列调整过的节点组来说非常有用。可通过细节选项卡来逐个为注释框进行此项操作的变更，如图3-96所示。

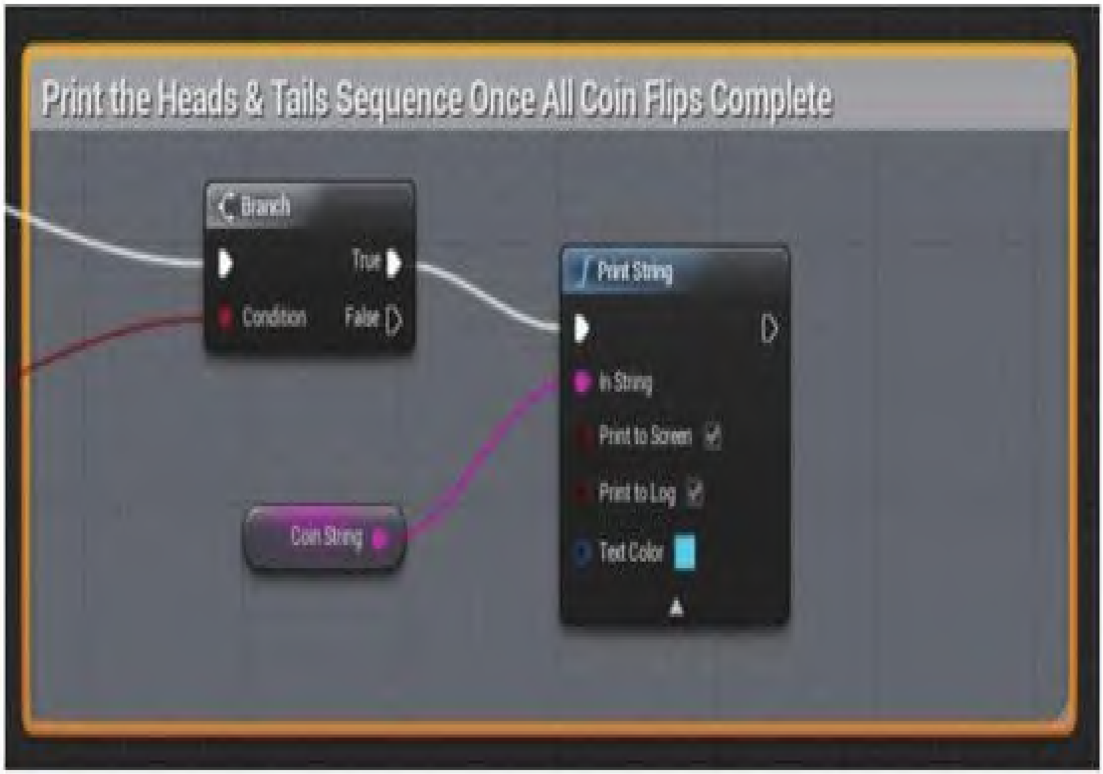



图3-96

请注意注释框会随着图表的放大缩小而缩放，所以把图表缩小后，大致看一眼注释框会对了解其功能很有帮助，如图3-97所示。

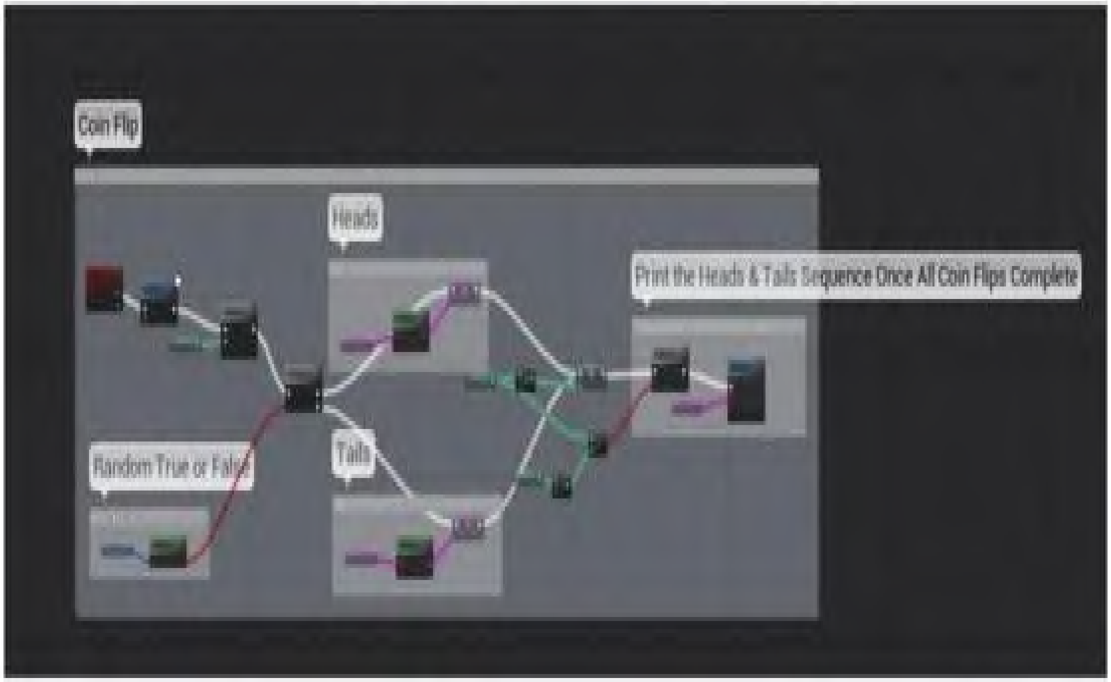



图3-97

### 2.编辑注释框文本

可以右击注释框的页眉位置，在右击出现的菜单中变更Node Comment（节点注释）文本框的内容来改变你注释的文本。按下回车来保存你的新注释，如图3-98所示。

或者：

•选中注释框后按下F2键来编辑注释。

•双击注释框的页眉位置来编辑注释。

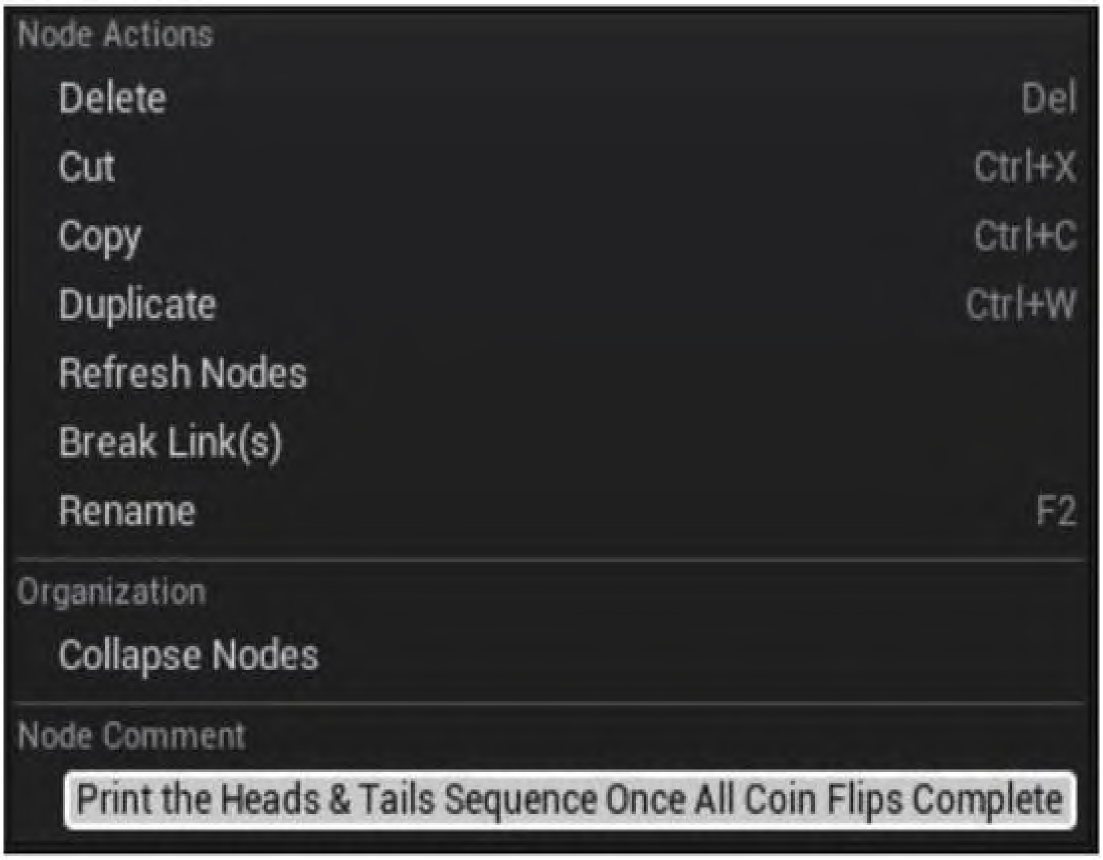



图3-98

### 3. 删除注释框

在任何注释框体上单击并按下删除键，或右击注释框的页眉位置，并从右击出现的菜单中选择删除。这些选项将会删除注释框，但对于注释框内包含的节点无效。

### 4. 注释框属性

注释框属性可以在细节选项卡中分别对每个注释框进行调整。只要单击注释框即可在细节选项卡中显示其属性，具体见下表。

| 项　目 | 描　述 |
| --- | --- |
| Properties | |
| 注释颜色 | 该属性可以让你变更注释框的背景颜色。请记住使用 Alpha 值来调整背景的透明度 |
| 对注释泡泡上色 | 如此项被勾选，则注释颜色将会被应用到注释框中 |
| 移动模式 | 该属性可让你设置在移动注释时是否要同时移动其中的节点，（如同时移动则为组移动模式），或移动注释时为单独移动，不移动其中的节点（注释模式） |

3.3

# 蓝图指导

如你刚接触到UE4的蓝图——可视化脚本，“蓝图指导”则大有裨益。它可提供蓝图使用核心要素的信息，包括创建蓝图、添加和连接节点、创建变量以及下表所示的其他内容。

## 3.3.1　创建蓝图类

**1.通过内容浏览器创建蓝图**

Blueprint（蓝图）存储在包中可以像创建任何其他资源一样通过内容浏览器来创建蓝图，如图3-99所示。

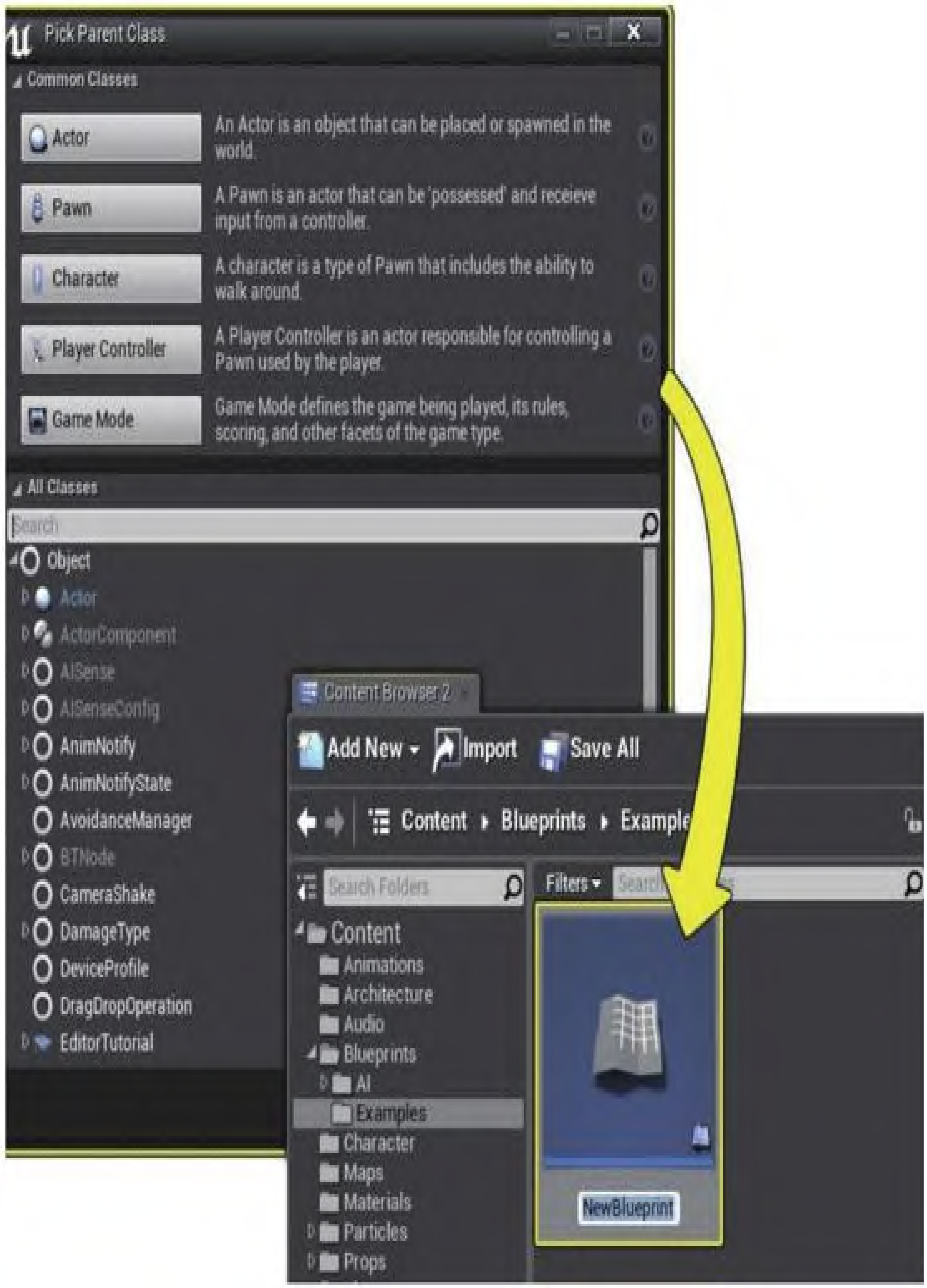



图3-99

**Step 1** 在Content Browse里面，单击 。

**Step 2** 在出现的菜单下面选择Blueprint。

额外的蓝图类型可以在Create Advanced Asset内的Blueprints选项中创建。

**Step 3** 为你的蓝图选择一个Parent Class，如图3-100所示。

图3-100

### 2. 右击创建

在Content Browser中创建Blueprint有两种方法。

**Step 1** 在Content Browser里的Asset View（right-hand panel）里面右击。

Step 2　在出现的菜单中选择新建Blueprint，如图3-101所示。

Step 3　为你的蓝图选择一个父类，如图3-102所示。

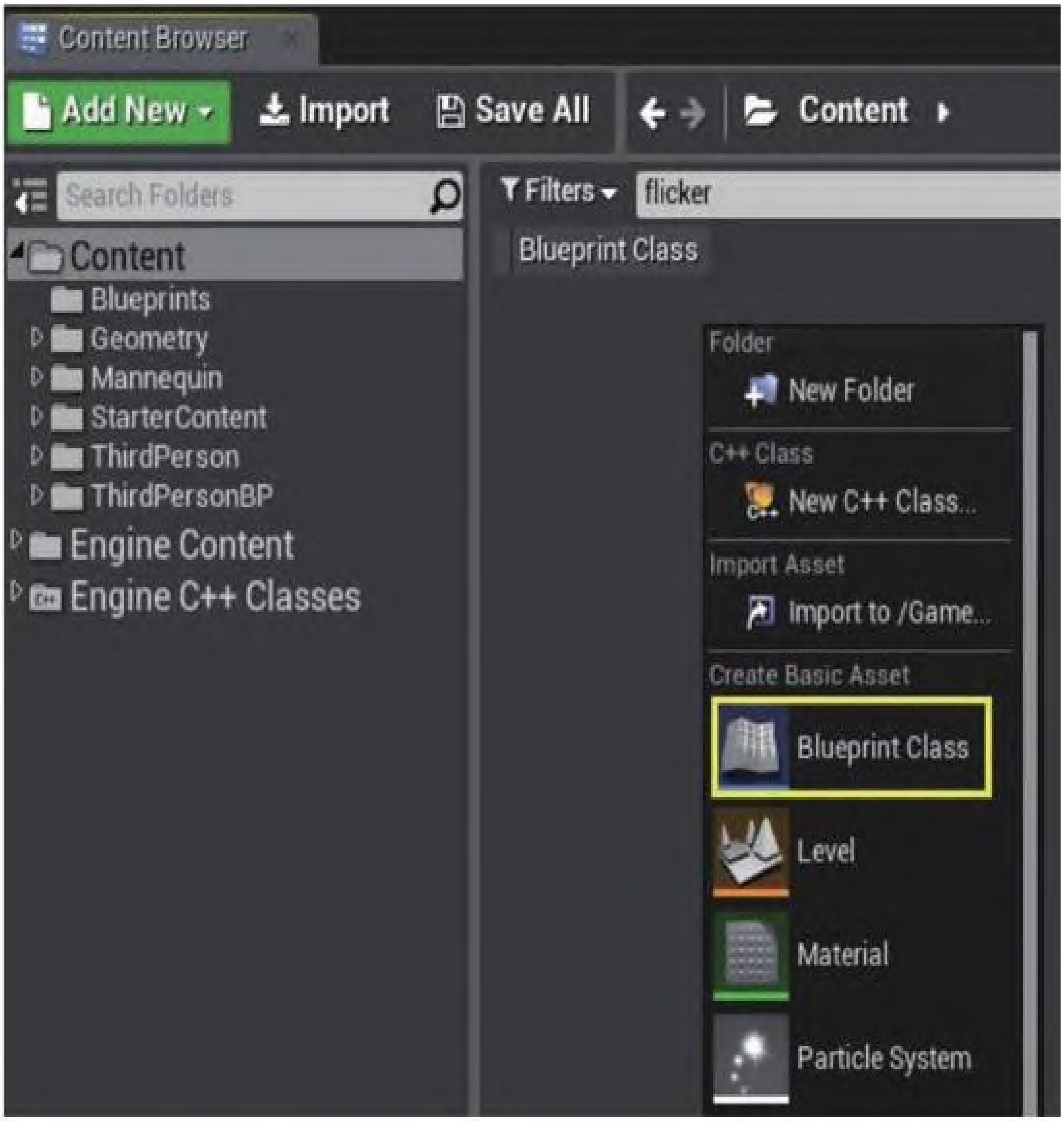



图3-101

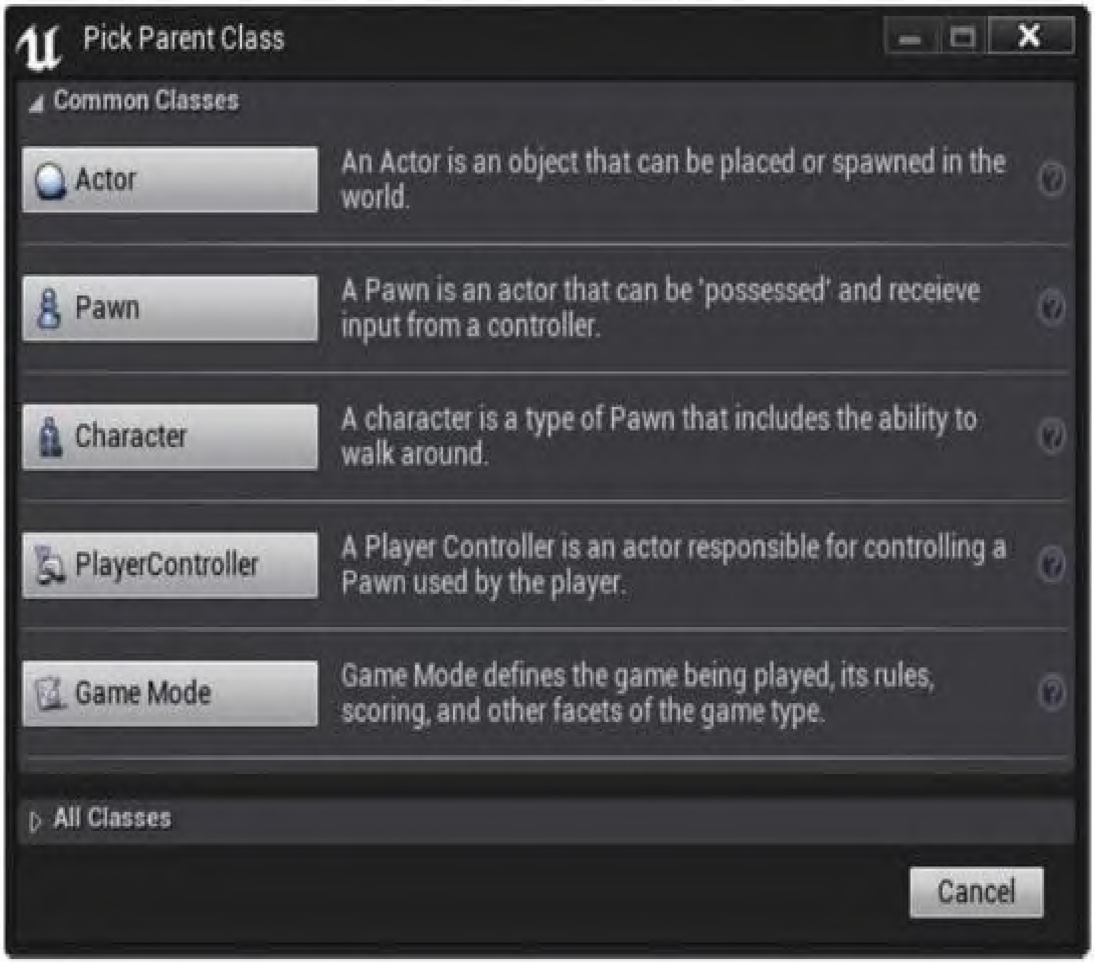



图3-102

或者：

**Step 1**　在Content Browser的Asset Tree（left-hand panel）上右击。

**Step 2**　在出现的菜单中，鼠标移到Create Asset上，然后选择新建Blueprint，如图3-103所示。

**Step 3**　为你的蓝图选择一个父类，如图3-104所示。

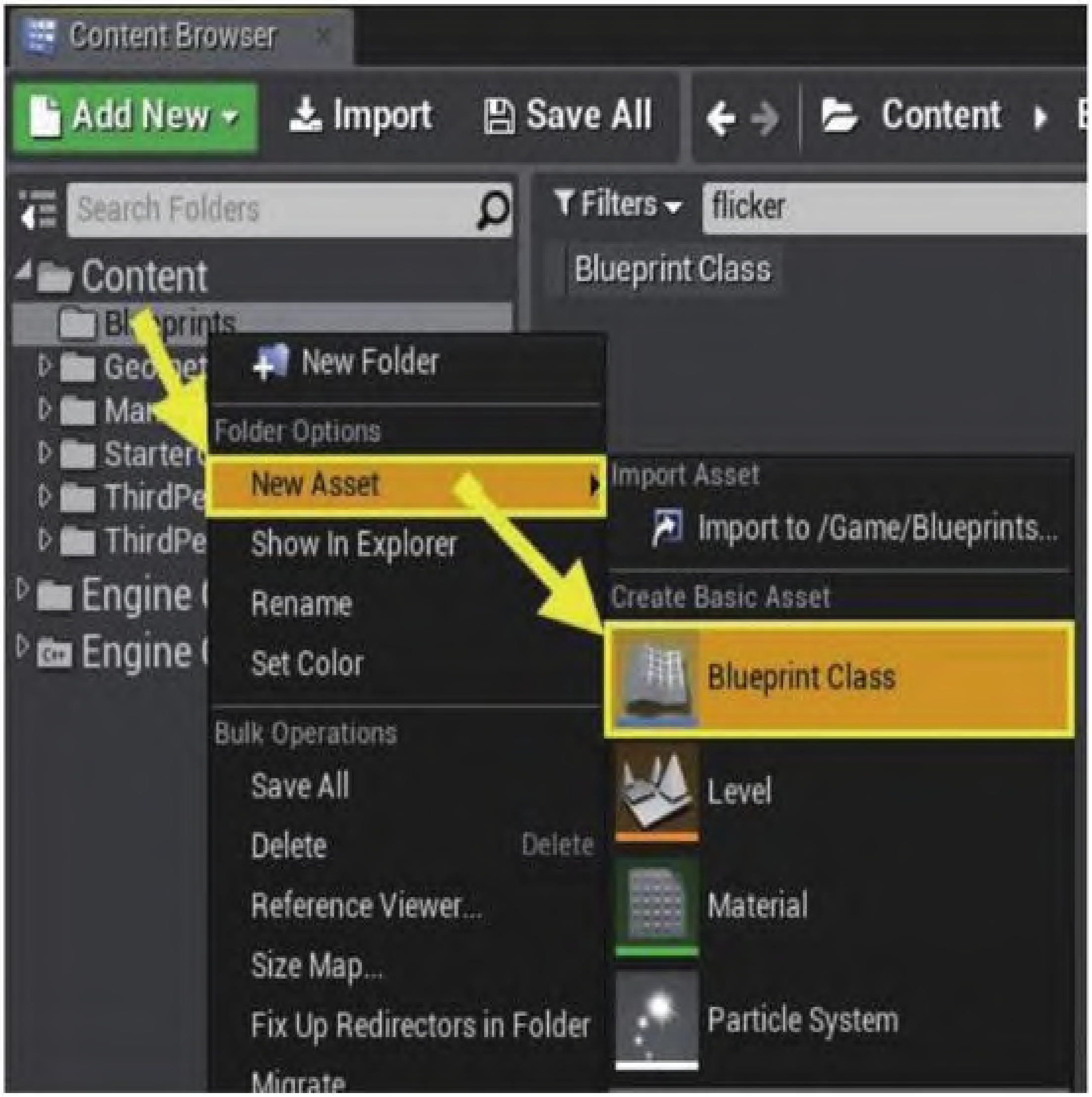

Content Browser
Add New
Import
Save All
Content
Search Folders
Filters
flicker
Blueprint Class
Content
New Folder
Folder Options
New Asset
Show In Explorer
Rename
Set Color
Bulk Operations
Save All
Delete
Delete
Reference Viewer...
Size Map...
Fix Up Redirectors in Folder
Import Asset
Import to /Game/Blueprints...
Create Basic Asset
Blueprint Class
Level
Material
Particle System

图3-103

图3-104

## 3.3.2　连接节点

本节讲述在蓝图中的图表上连接节点的多种方法。

### 3.3.2.1　引脚至引脚连接

连接节点的最常用方法为引脚至引脚连接。

使用鼠标左键拖动一个引脚到另一个兼容的引脚上，如图3-105所示。

鼠标悬停在一个兼容引脚上时将出现一个绿色的小钩。

尝试连接两个不兼容的引脚时，将出现图标提示节点无法连接的原因，如图3-106所示。

图3-105

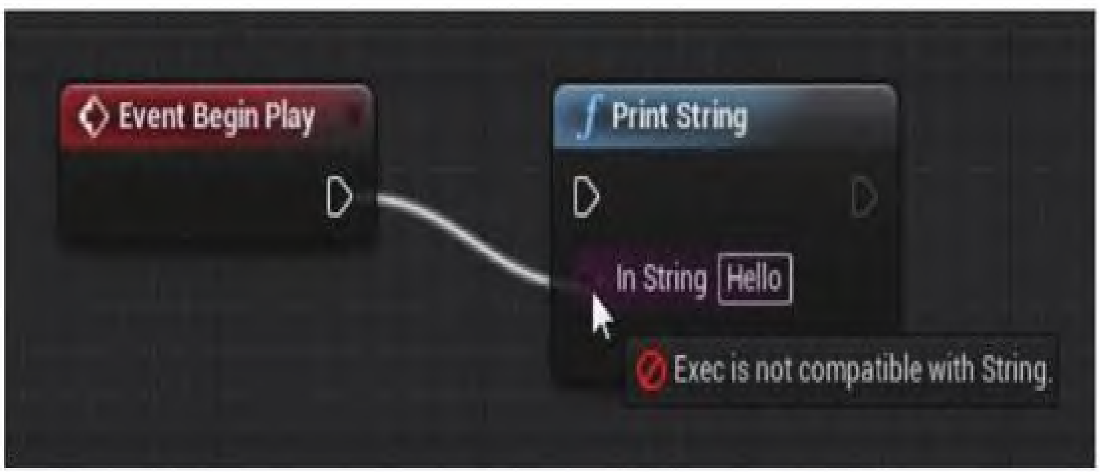



图3-106

引脚拥有典型的颜色编码，可以反映出它们接收的连接类型，如图3-107所示。

也存在连接两个不同类型引脚的情况，此时将创建一个转换节点，如图3-108所示。

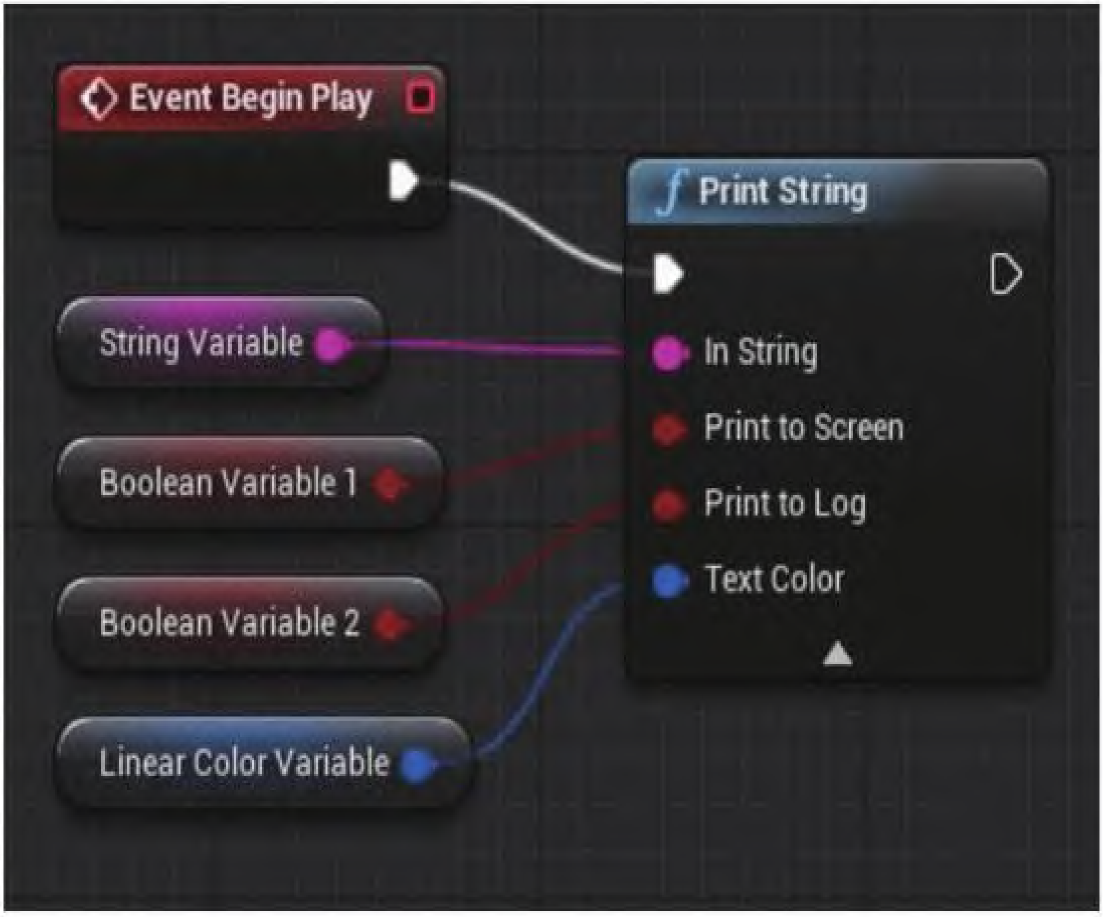

Event Begin Play
Print String
String Variable
In String
Print to Screen
Boolean Variable 1
Print to Log
Boolean Variable 2
Text Color
Linear Color Variable

图3-107

图3-108

在上图中，我们将一个Float输出和一个String输入连接起来。可以从提示文本中看到它们即将被转换。

相连之后，两个引脚之间加入了一个新的转换节点，如图3-109所示。

将引脚拖至图表中的空白处，以便放置新节点，如图3-110所示。

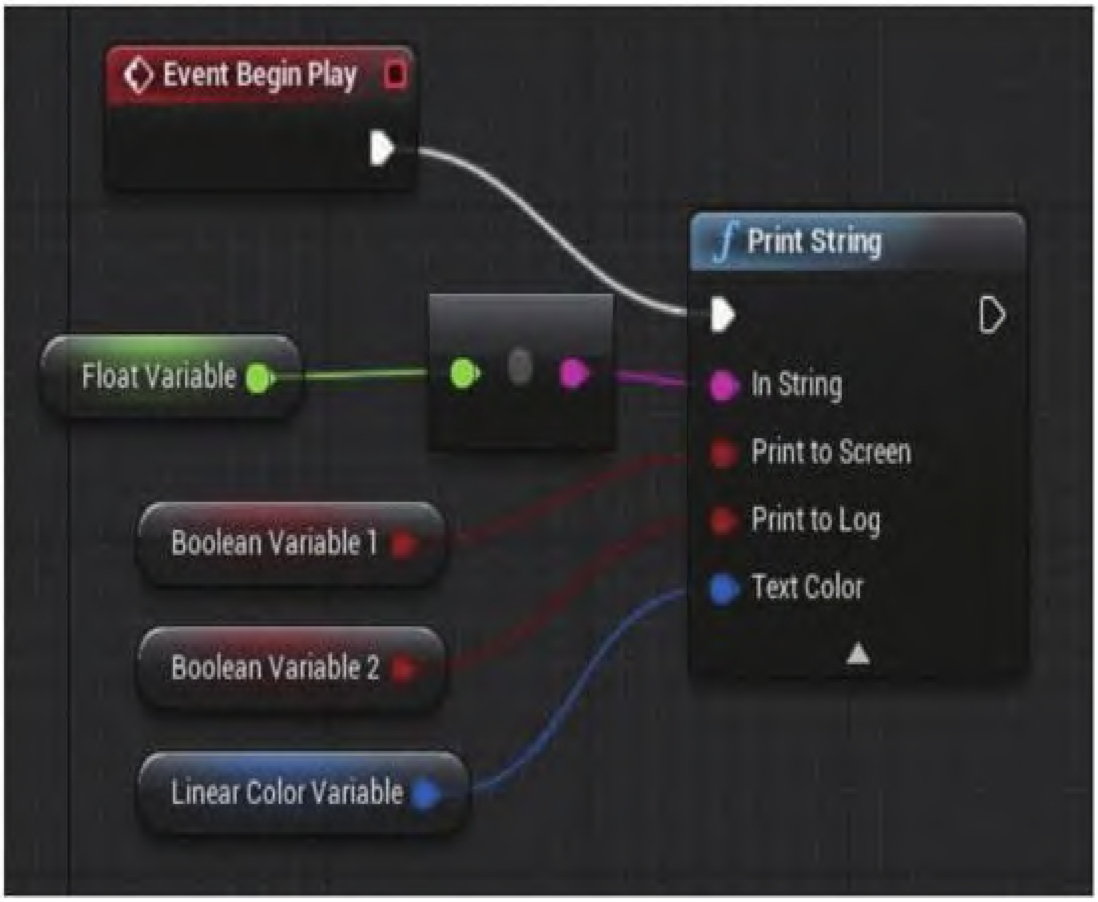



图3-109

图3-110

松开鼠标左键后将出现快捷菜单，如图3-111所示。

图3-111

选择需要放置的节点后，节点将自动连接。

也可选取现有连接，将其连接到另一兼容的引脚上，如图3-112所示。

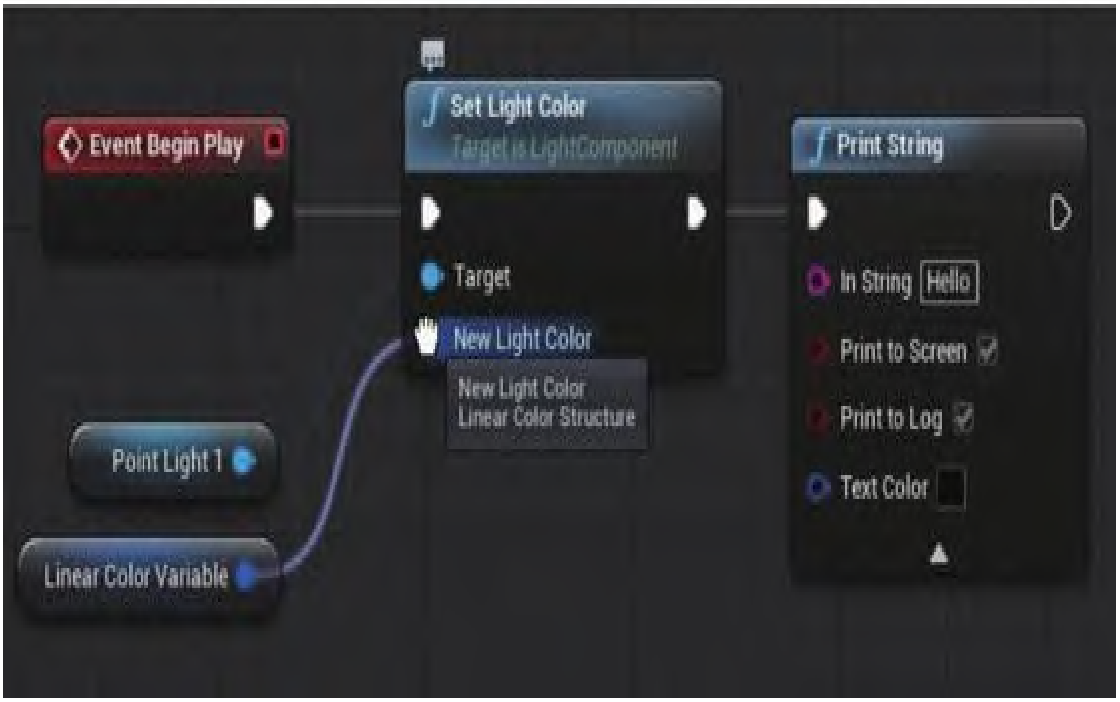



图3-112

在上图示例中，长按Ctrl键，单击引脚，将其拖至另一输入，如图3-113所示。

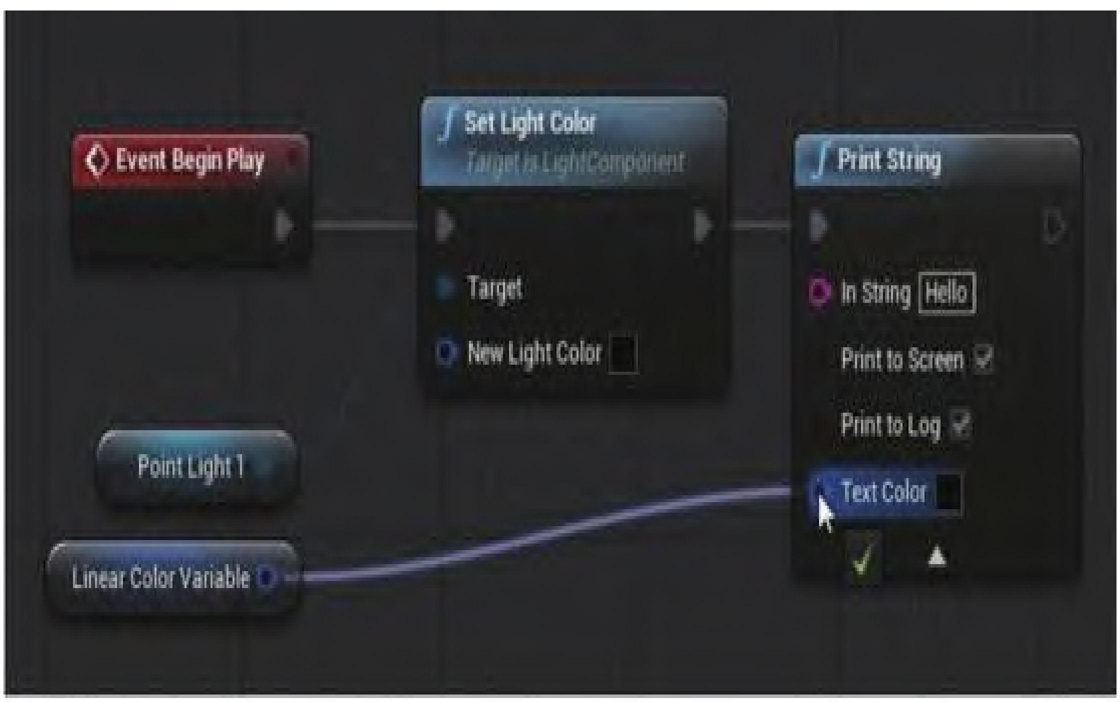



图3-113

此操作将把连接移至另一引脚而无须重新连接。

### 1. 引脚操作

一些额外引脚相关操作可用于节点，见下表和图3-114。

图3-114

| 操　作 | 命　令 |
| --- | --- |
| 引脚相关的关联菜单 | 右击引脚 |
| 着重显示连接线 | 鼠标悬停到引脚上 |
| 连接到另一个引脚 | 单击 + 拖动到那个引脚 |
| 引脚的过滤后的操作菜单 | 单击 + 拖动到图表 |
| 断开所有连接 | Alt+ 单击引脚 |
| 移动所有连接 | Ctrl+ 鼠标左键拖动到引脚 |

## 2.节点至节点连接

使用变量时可选择一些节点至节点连接，以节约时间。

举例来说，脚本中存在一个已连接的Set变量节点（在此节点中设置变量值），而你希望在无须重新连接脚本的前提下将其与另一变量交换。可拖动另一变量到希望改变的变量上，其将被自动切换并保持连接不断开，如图3-115所示。

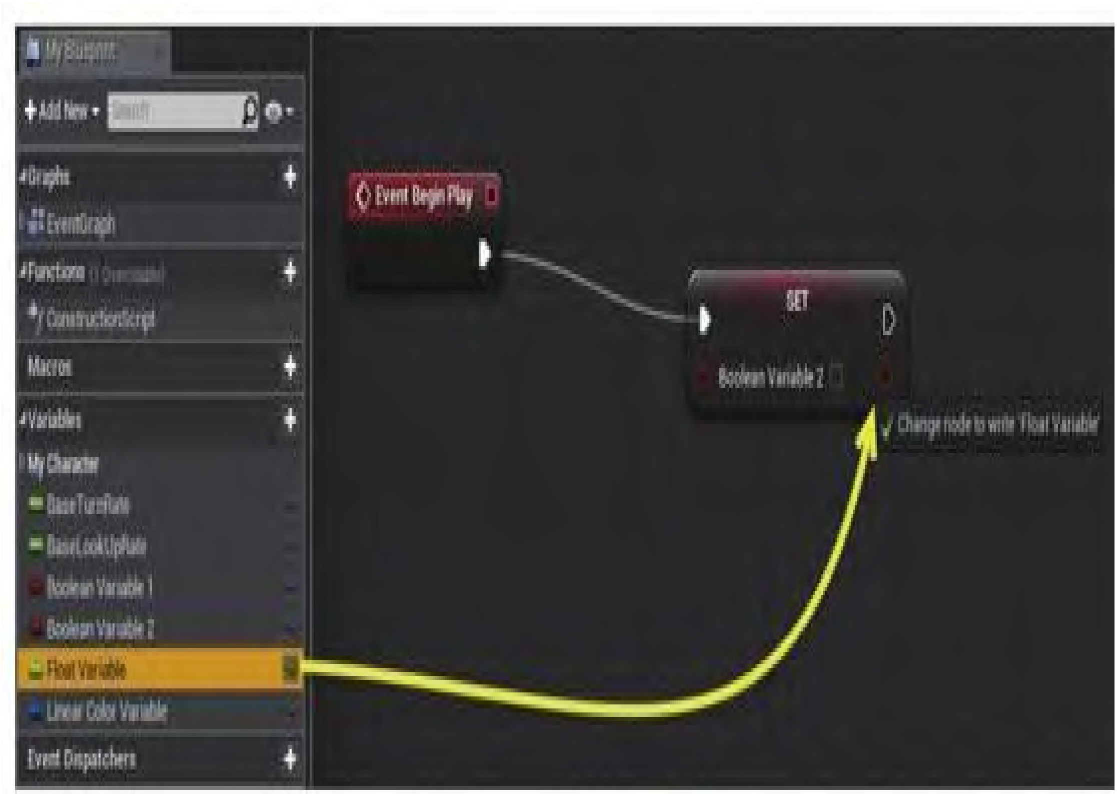



图3-115

如上图所示，Event Begin Play节点将Boolean Variable 2设为false，需要将它换为我们创建的Float节点。拖动Float节点到Set变量上，提示文本将告知你正在执行的操作将使节点发生变化，将对Float变量进行写入或设置，如图3-116所示。

图3-116

确认执行此操作后，节点将切换为新节点，连接也不会断开。

可对已连接的Get节点执行相同操作（下图示例中正在更换Bool变量），如图3-117所示。

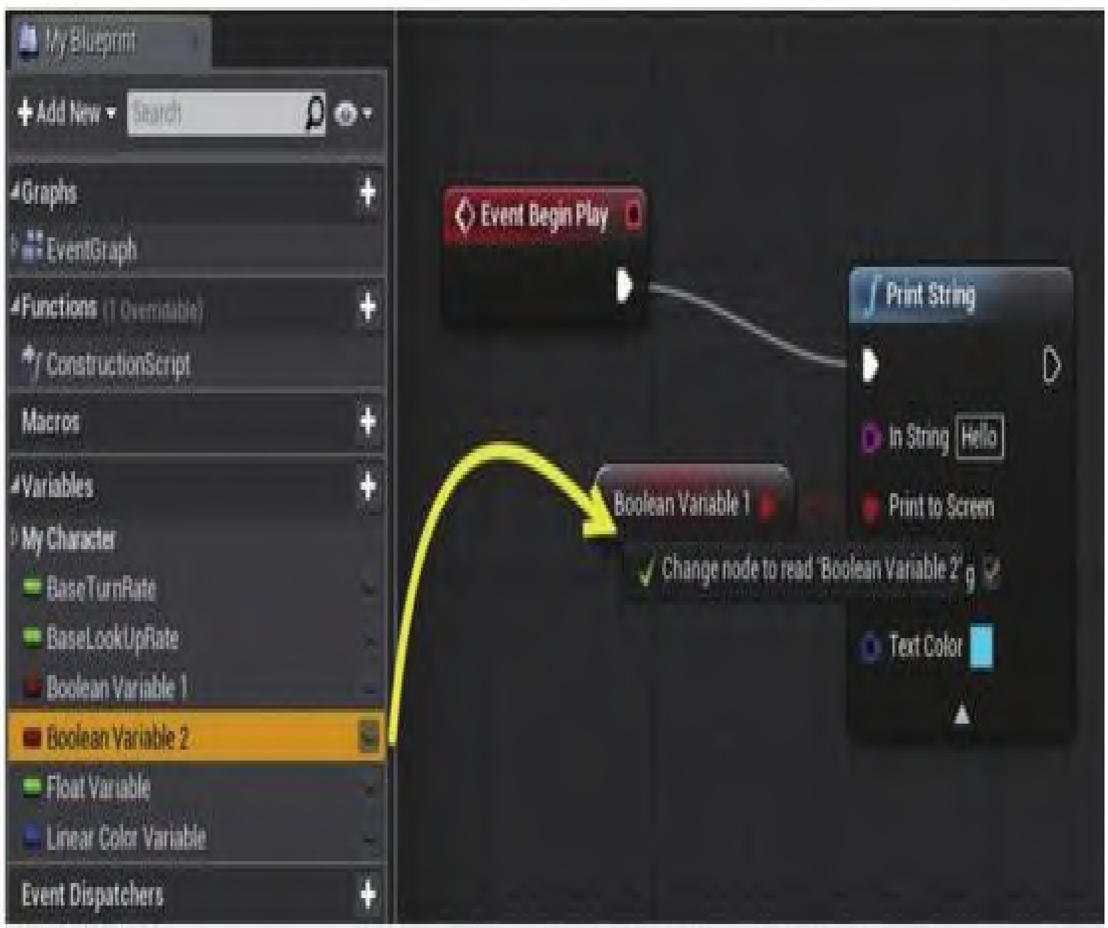



图3-117

### 3.3.2.2　拖放连接

使用变量的另一个捷径是执行拖放（节点至引脚）连接。

在下图示例中，需要在Event Begin Play节点上设置Point Light的light color。Set Light Color节点询问New Light Color，而我们已为其创建了一个Linear Color变量。在MyBlueprint窗口中，直接将该变量拖放至图表中的引脚上，如图3-118所示。

图3-118

执行此操作后，变量自动与节点相连，如图3-119所示。

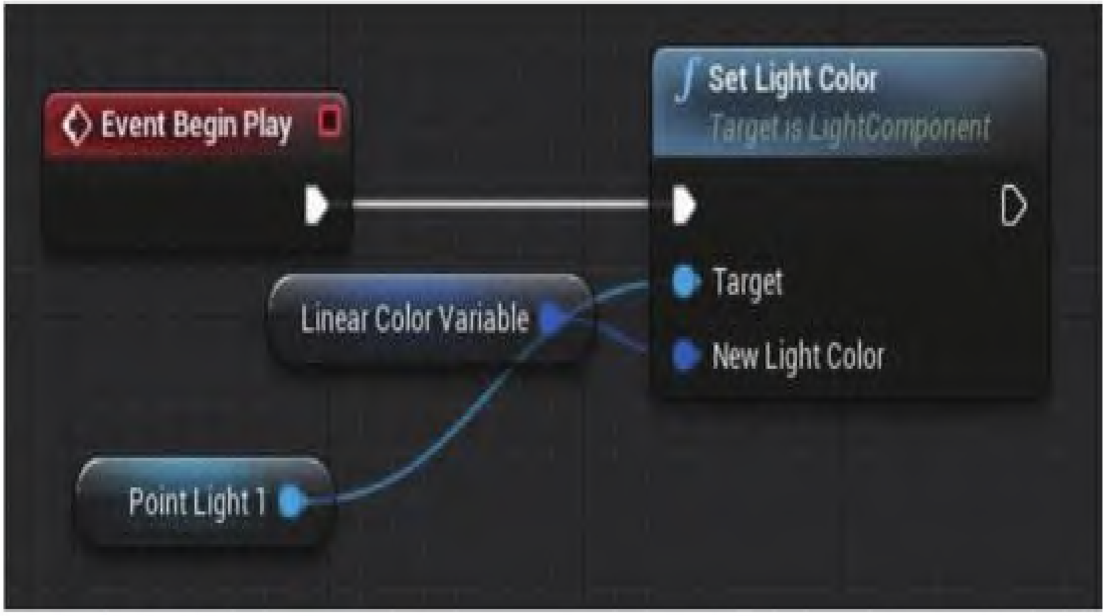



图3-119

此方法可避免添加节点至图表，直接执行引脚至引脚的连接。

### 3.3.2.3　重设连接路线

使用复杂或大型蓝图时，可能会出现连线杂乱无章的状况，需要在图表中改变其显示方式，以便清楚可见。可利用选项添加一个Reroute节点，新增一个“延长线”类型到输出引脚，以便移动线路。

在下图示例中，输出引脚的线路隐藏在黄色框中的节点后面，如图3-120所示。

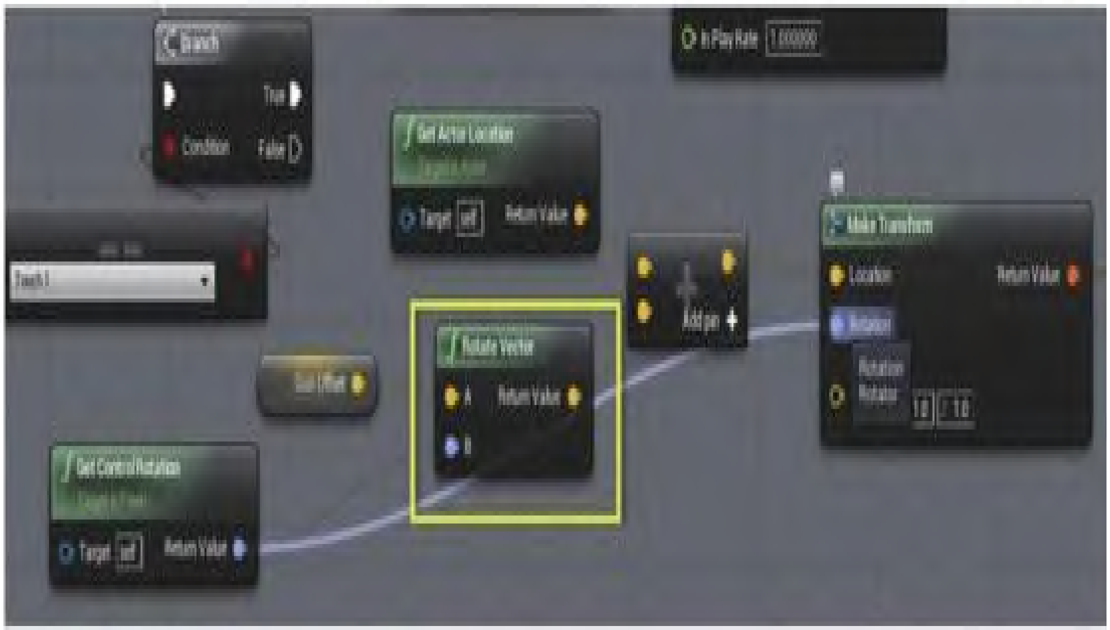



图3-120

可拖动希望重设线路的输出引脚，选择Add Reroute Node选项，如图3-121所示。

图3-121

此操作将在旧引脚连接的图表中创建一个新引脚，如图3-122所示。

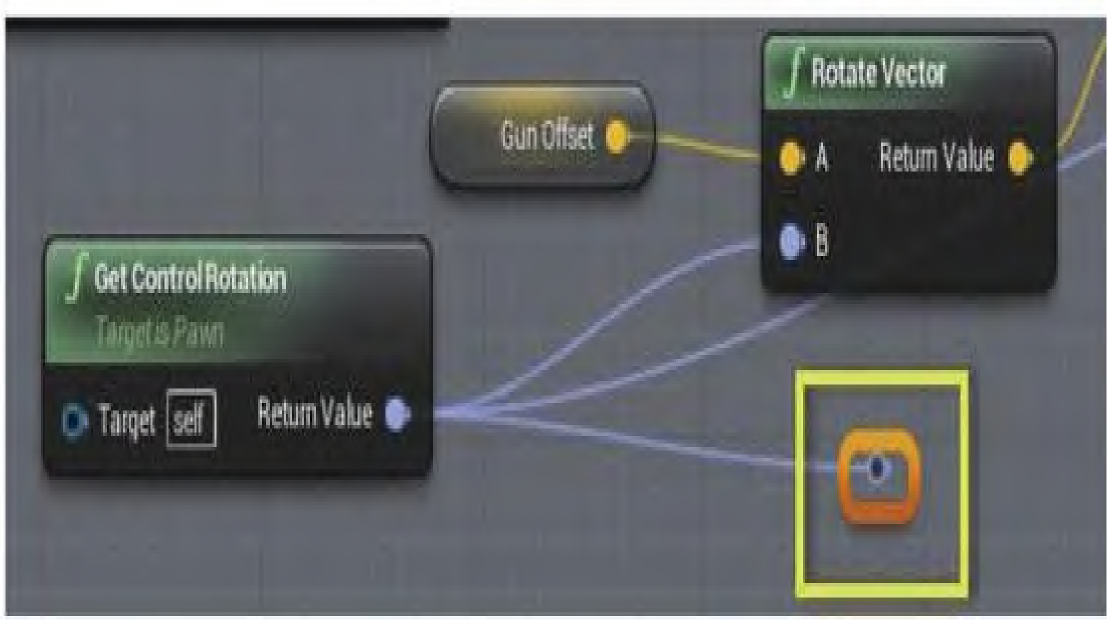



图3-122

然后拖动新引脚并将其和目标连接起来，如图3-123所示。

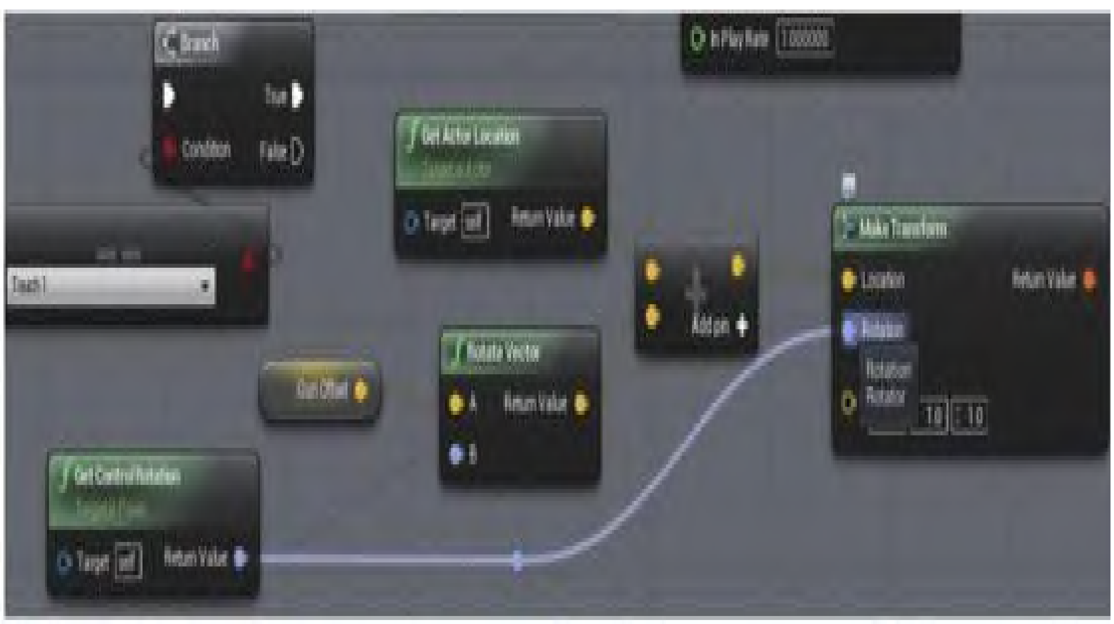



图3-123

Reroute节点作为旧引脚，可被拖离。可将新节点和它连接起来，如图3-124所示。

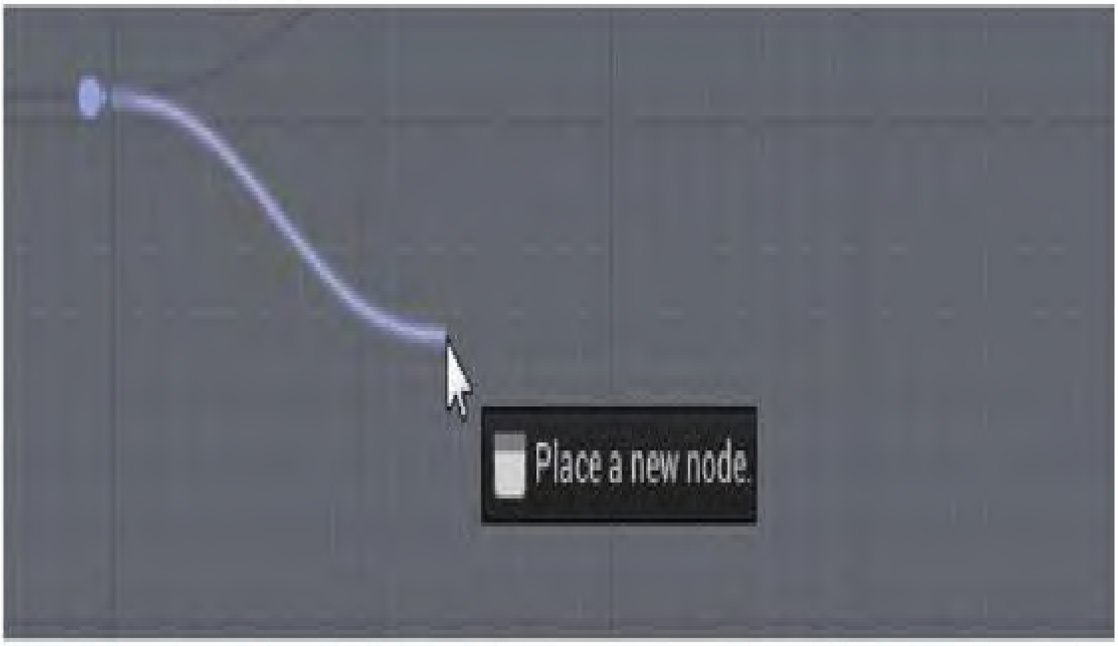



图3-124

利用Reroute节点可使蓝图整洁美观，使访问者对连接一目了然。

## 3.3.3 重叠图表

本节讲述将Blueprint图表重叠为一个单独节点、一个函数或一个宏的多种方法。可以将大型图表重叠为一个连接图表，便于清理。

### 3.3.3.1 重叠节点

下面的示例应用的是最简单的重叠方法，将重叠节点转变为连接图表。它主要用于组织，可以使事件图表中的内容整齐清楚。

**Step 1** 在蓝图的图表上找到需要重叠的节点，如图3-125所示。

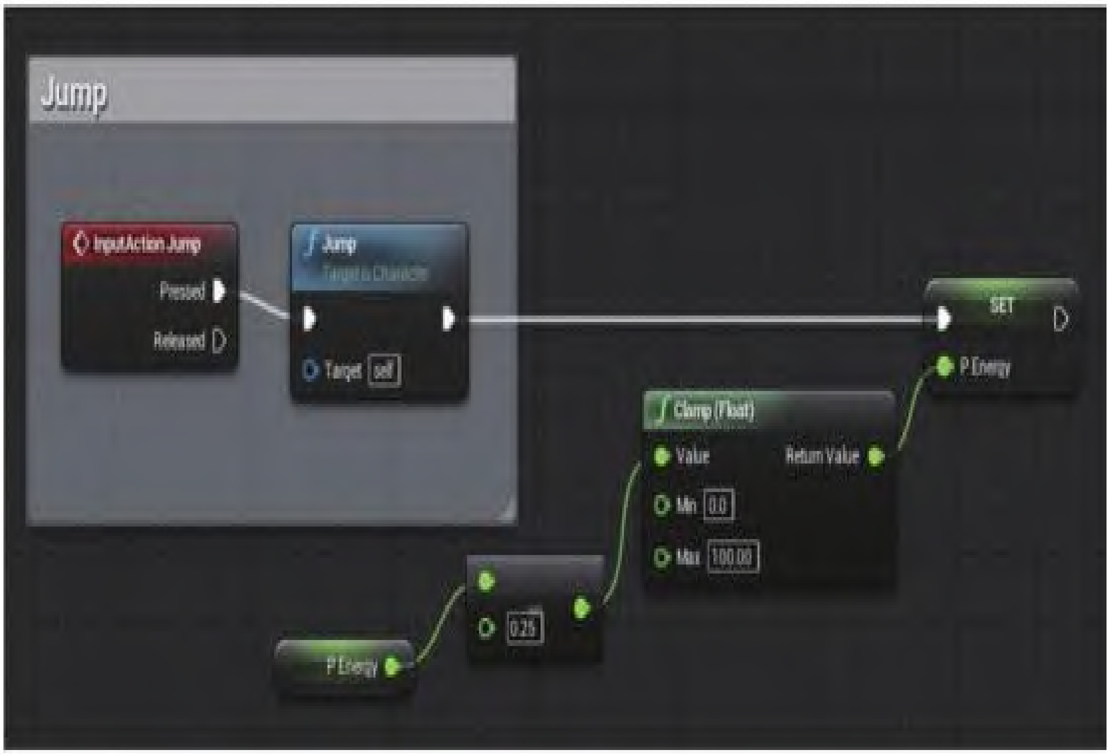



图3-125

此处有一些在玩家跳跃后减少其能量的节点。

**Step 2**　在节点上单击并拖动，选择需要重叠的所有节点，如图3-126所示。

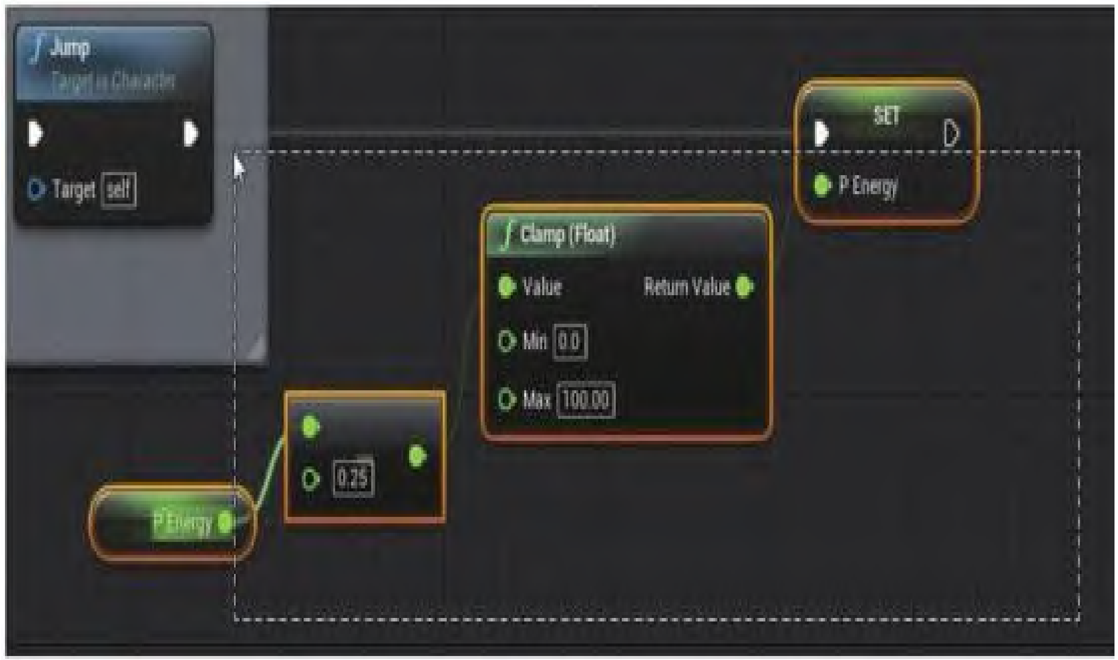



图3-126

可通过Ctrl+左键单击组合键从选择中添加/减去节点。

**Step 3**　选定需要重叠的节点后，在选中的任意节点上右击并选择Collapse Nodes，如图3-127所示。

**Step 4**　所有节点将重叠为一个单独的Collapsed Graph节点，然后可为该节点命名，如图3-128所示。

**Step 5**　将鼠标悬停在Collapsed Graph节点上，可以看到节点网络的预览窗口，如图3-129所示。

SET
P Energy
Clamp (Float)
Value
Min 0.0
Max 100.00
Node Actions
Delete Delete
Cut Ctrl+X
Copy Ctrl+C
Duplicate Ctrl+W
Refresh Nodes
Break Link(s)
Rename F2
Organization
Collapse Nodes
Collapse to Function
Collapse to Macro
Comment Group
Create Comment from Selection
Collapses selected nodes into a single node

图3-127

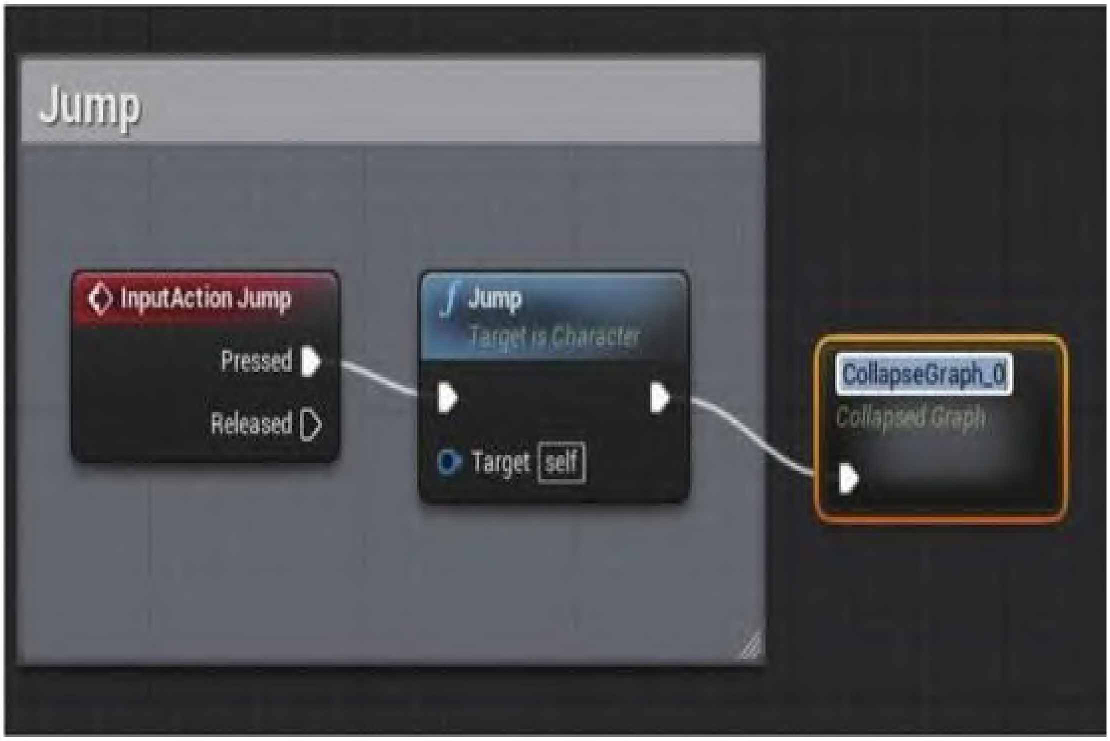

Jump
InputAction Jump
Pressed
Released
Jump
Target is Character
Target self
CollapseGraph_0
Collapsed Graph

图3-128

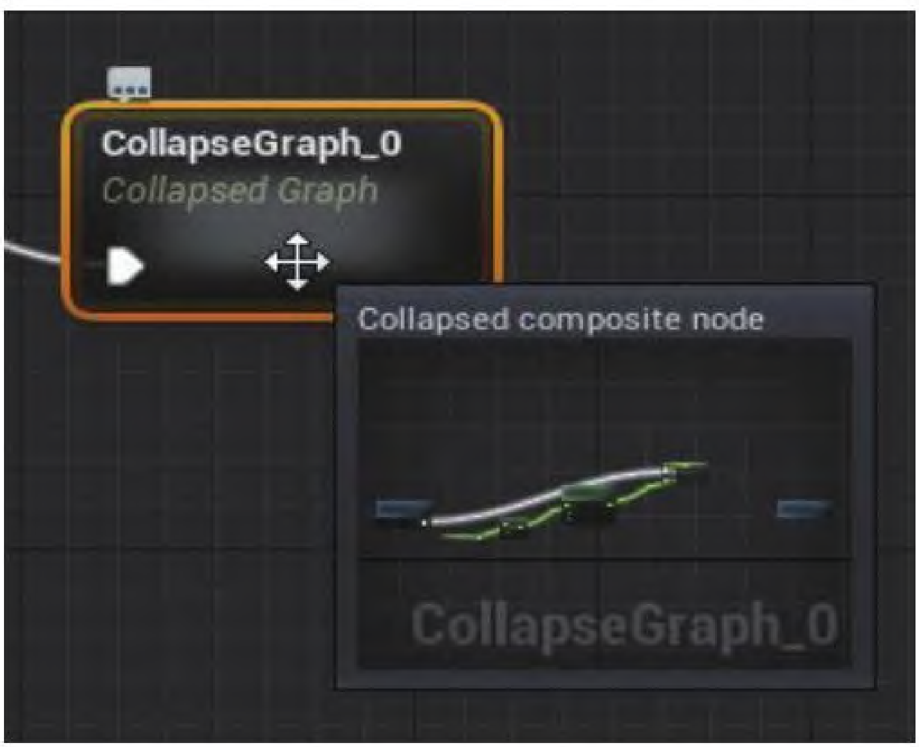

CollapseGraph_0
Collapsed Graph
Collapsed composite node
CollapseGraph_0

图3-129

**Step 6**　双击Collapsed Graph节点将打开含所选节点的新图表，如图3-130所示。

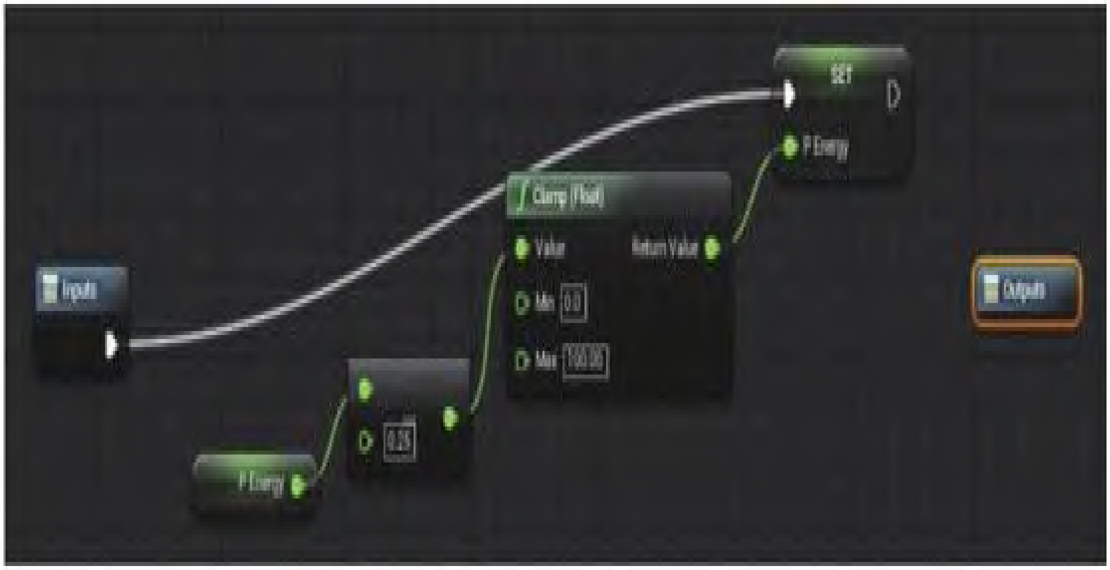



图3-130

**Step 7**　在Collapsed Graph中，可从Details面板添加引脚至Inputs或Outputs节点，如图3-131所示。

图3-131

重叠时将默认添加Execute，可在此添加两个Execute输出引脚：OutOfEnergy和CanJump。

我们也对图表进行了修改，在扣除玩家拥有的能量之前确定能量是否已经耗尽，如图3-132所示。

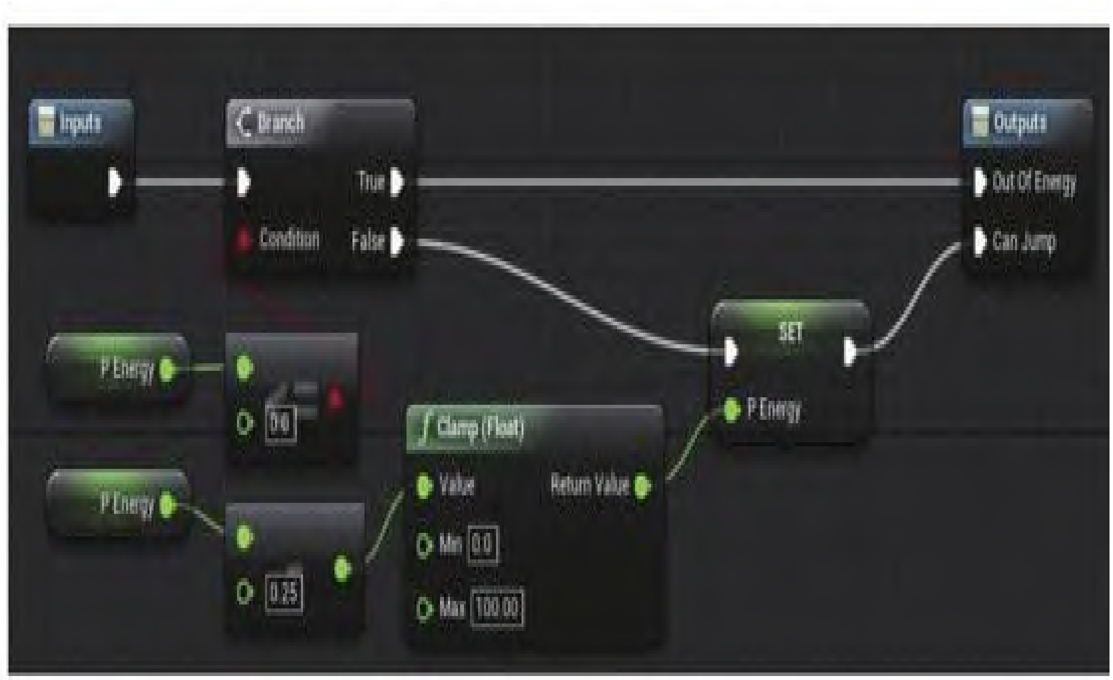



图3-132

带重叠节点的事件图表外观如图3-133所示。

**Step 8** 在重叠节点上右击，可选择Expand Node将节点转换为其原始状态，如图3-134所示。

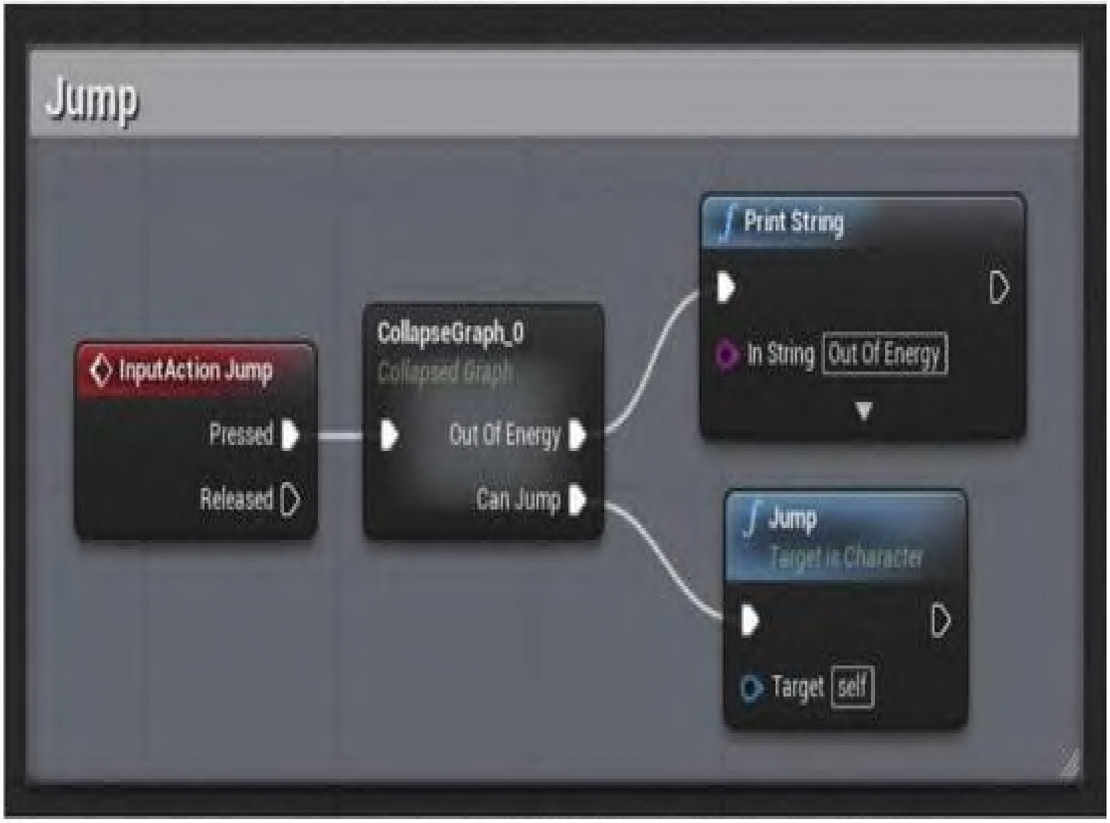

Jump
Print String
In String
Out Of Energy
InputAction Jump
CollapseGraph_0
Collapsed Graph
Pressed
Out Of Energy
Released
Can Jump
Jump
Target is Character
Target
self

图3-133

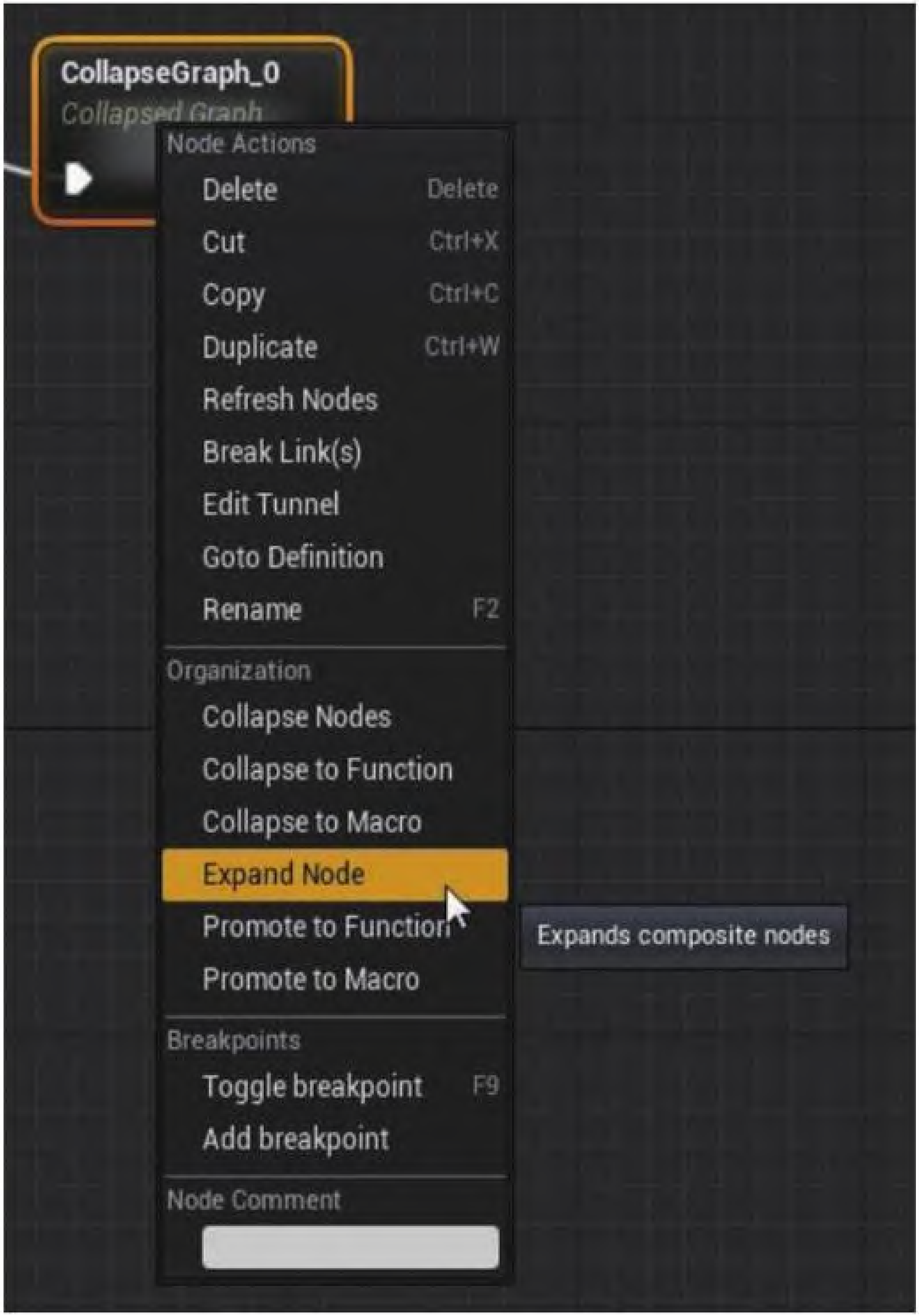



图3-134

### 3.3.3.2　重叠为函数

重叠节点的另一种方式是将它们重叠为一个函数。较之于重叠图表，将节点重叠为函数有下列优点：基于任意指定参数在脚本的另一区域调用函数、在另一蓝图中调用函数（需为包含要调用函数的蓝图添加引用）。

下列步骤说明如何将节点重叠为函数。

**Step 1** 选中需要重叠的节点，在其中一个节点上右击并选择Collapse to Function，如图3-135所示。

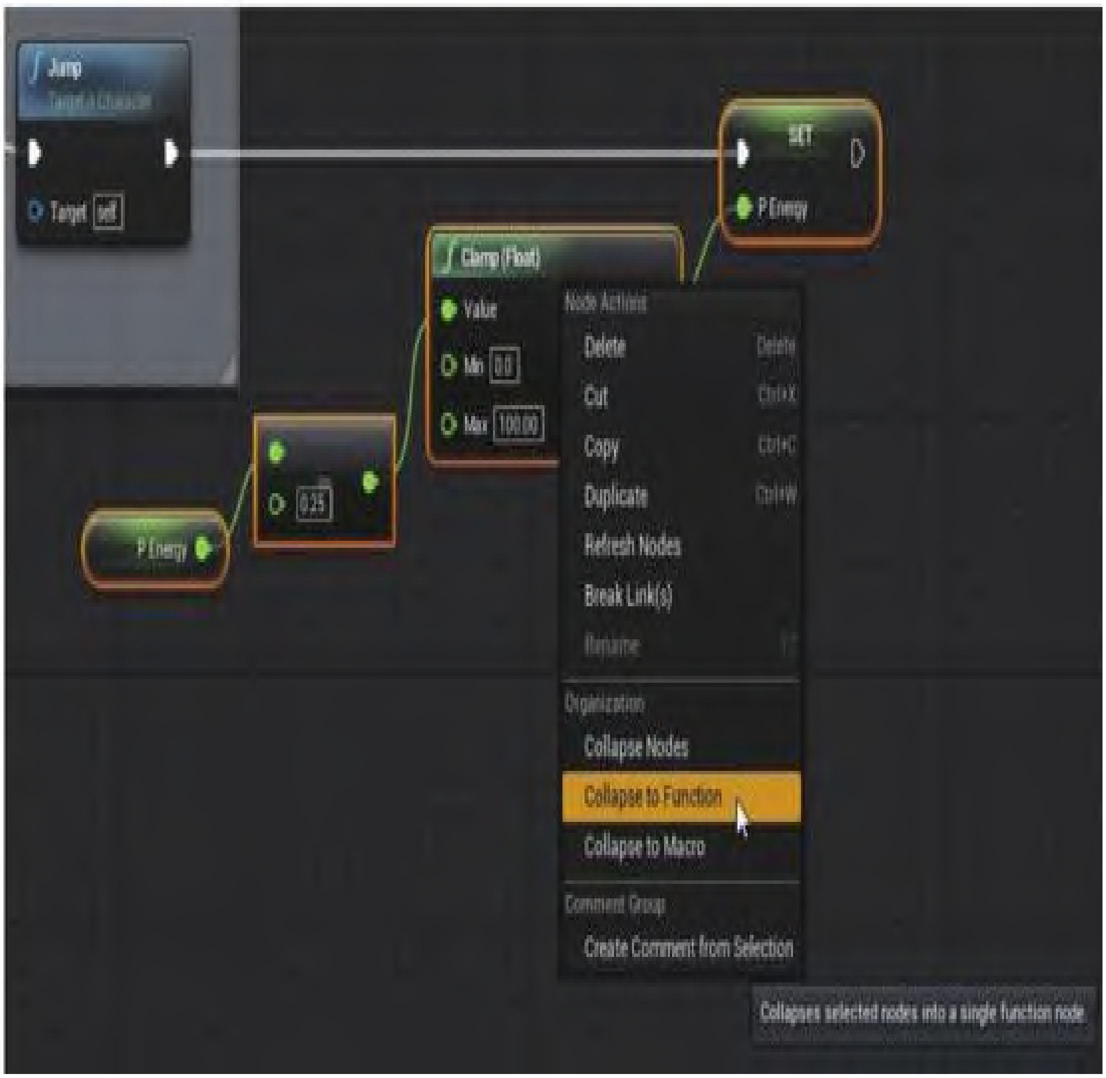



图3-135

**Step 2** 一个新Function将被创建，可在MyBlueprint窗口按F2键对其进行重命名，如图3-136所示。

编译蓝图后，“WARNING ！”报错信息将自动消失，如图3-137所示。

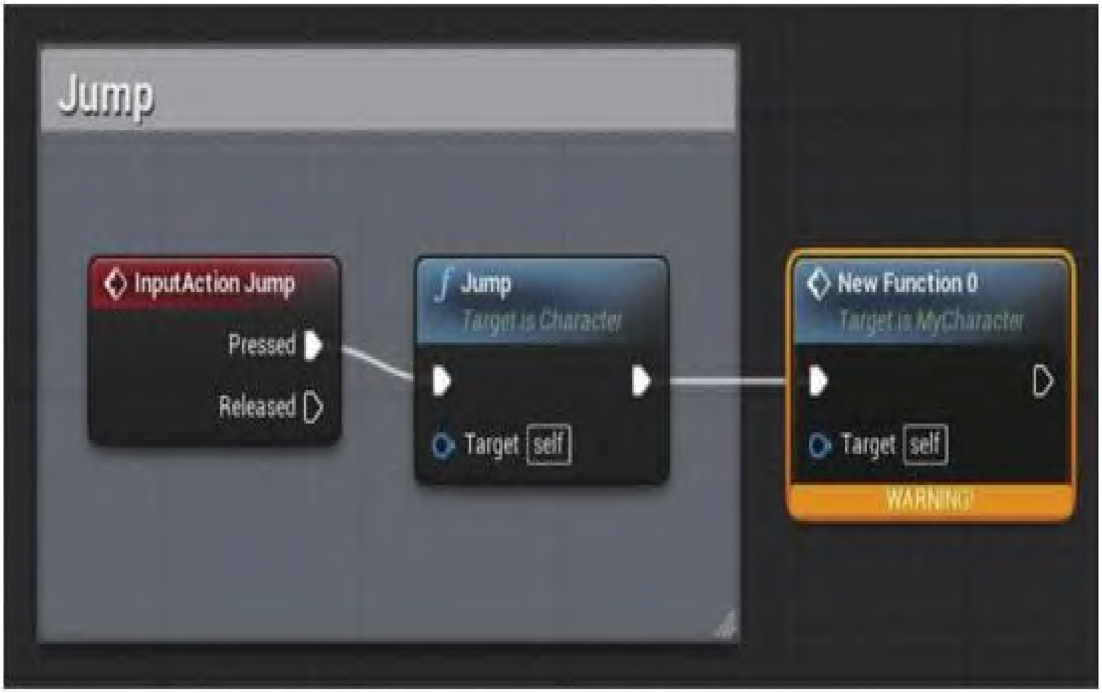

Jump
InputAction Jump
Pressed
Released
Jump
Target is Character
Target
self
New Function 0
Target is MyCharacter
Target
self
WARNING!

图3-136

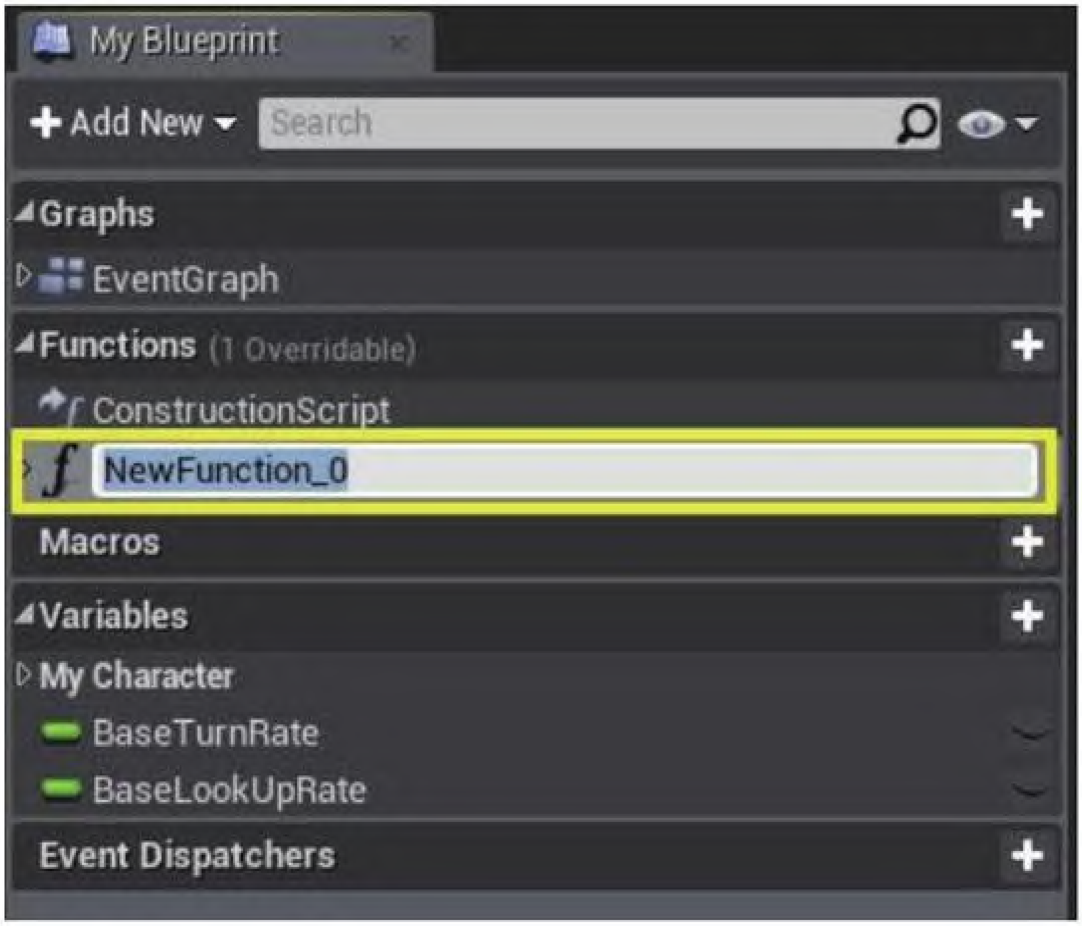



图3-137

在此处可对函数进行重命名。

Step 3　双击函数将其在一个新图表窗口中打开，如图3-138所示。

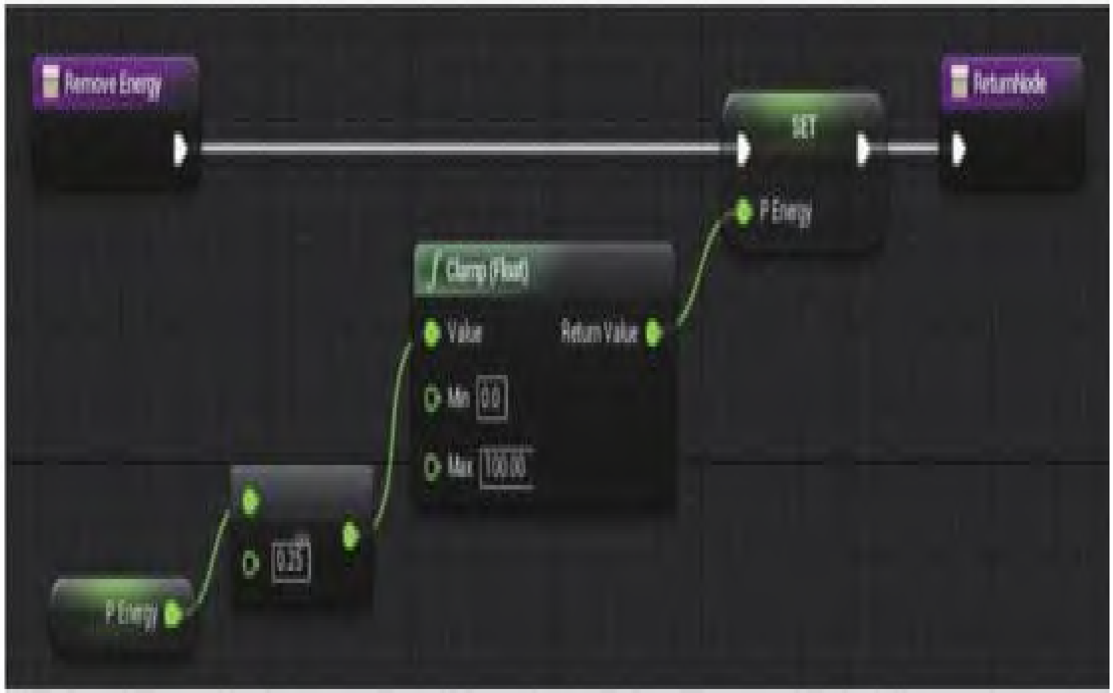



图3-138

和重叠图表相似，可单击Inputs或Outputs节点，通过Details面板为其添加引脚。

Step 4　可通过函数名指定其在何时被调用，如图3-139所示。

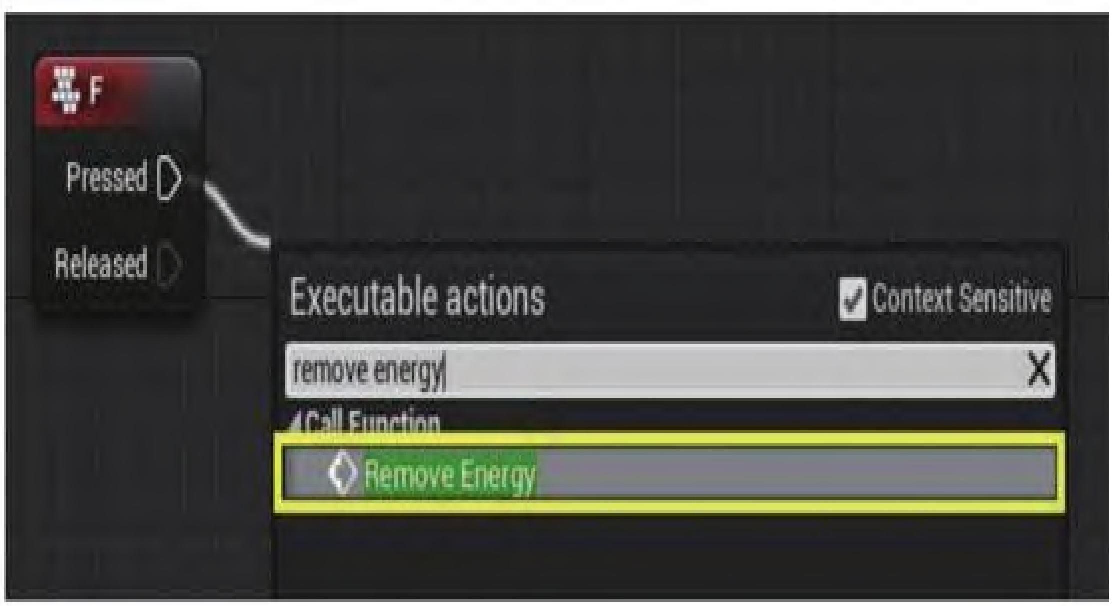



图3-139

在此处，F键被按下时将调用Remove Energy函数。

Step 5　在函数上右击并选择Expand Node可恢复重叠前的原始状态。

### 3.3.3.3　重叠为宏

除重叠为重叠图表和函数外，还可将节点重叠为宏。如此理解宏方为最佳：选取相连的数个节点，将它们合并为单个节点，以便于执行整体操作。利用其执行高频率操作（如复杂数学方程式）十分实用。

下列步骤讲述如何将节点重叠为宏。

Step 1　选中需要重叠的节点，在其中一个节点上右击并选择Collapse to Macro，如图3-140所示。

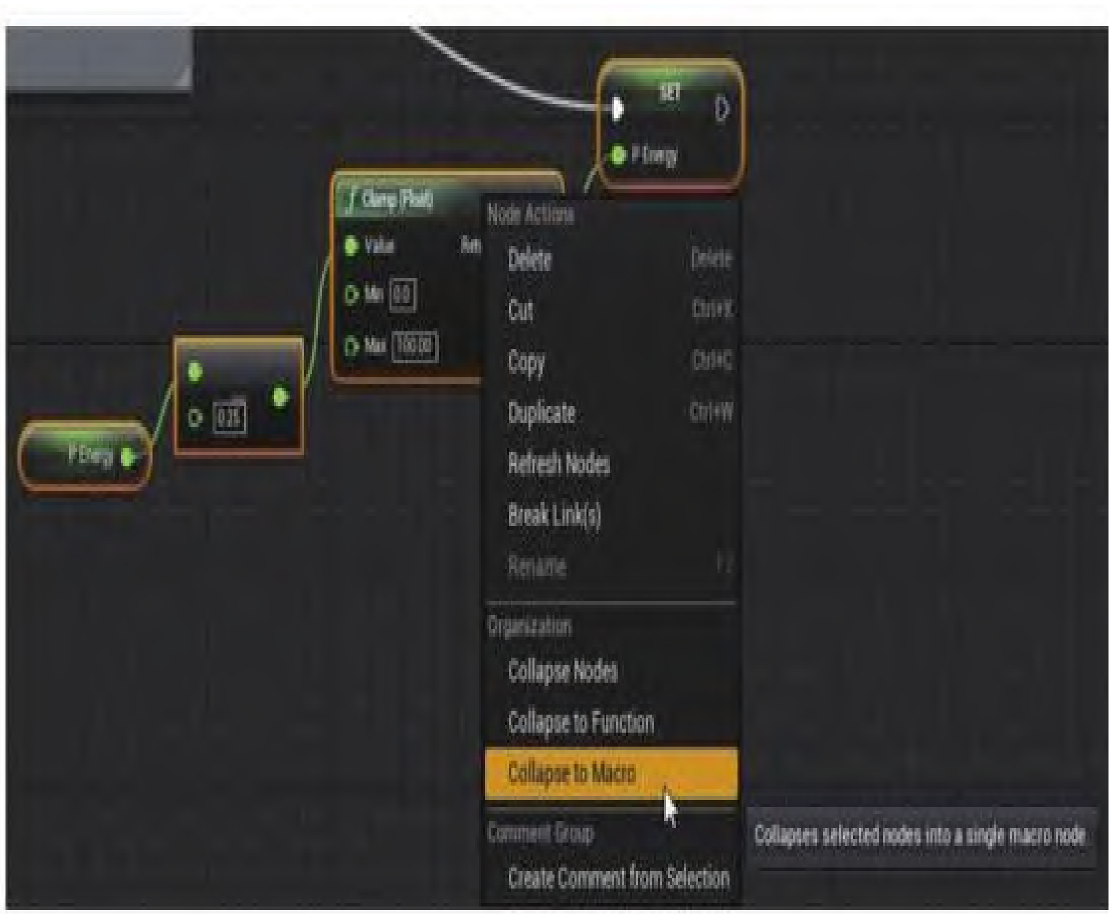



图3-140

Step 2　这样一个新宏节点就创建成功，可在MyBlueprint窗口按F2键对其进行重命名，如图3-141所示。

利用此方法可随意添加输入或输出引脚。

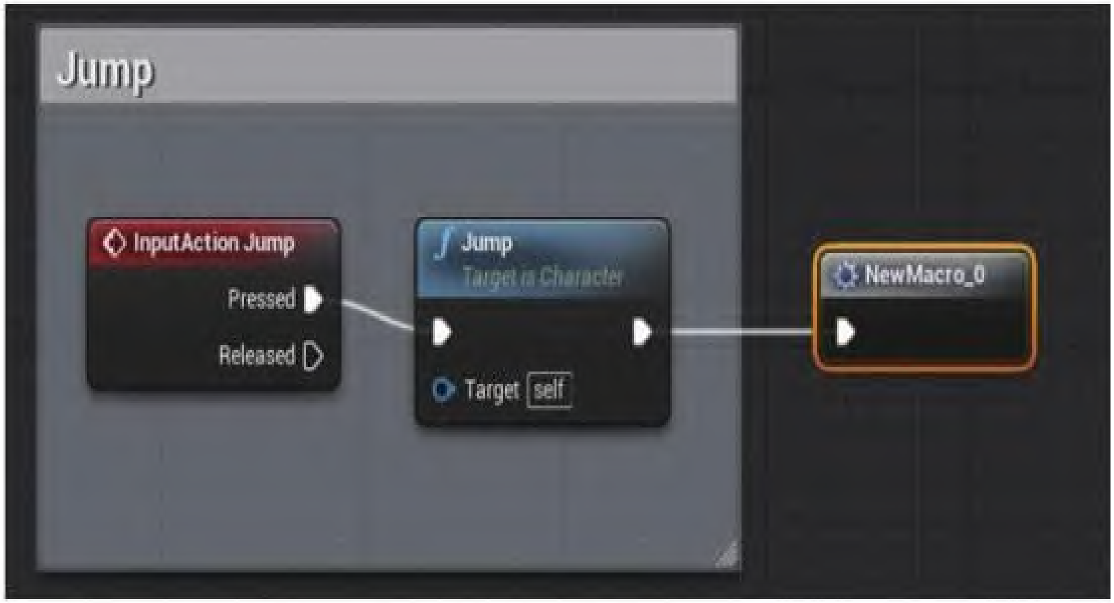



图3-141

Step 3　双击可将宏打开，如图3-142所示。

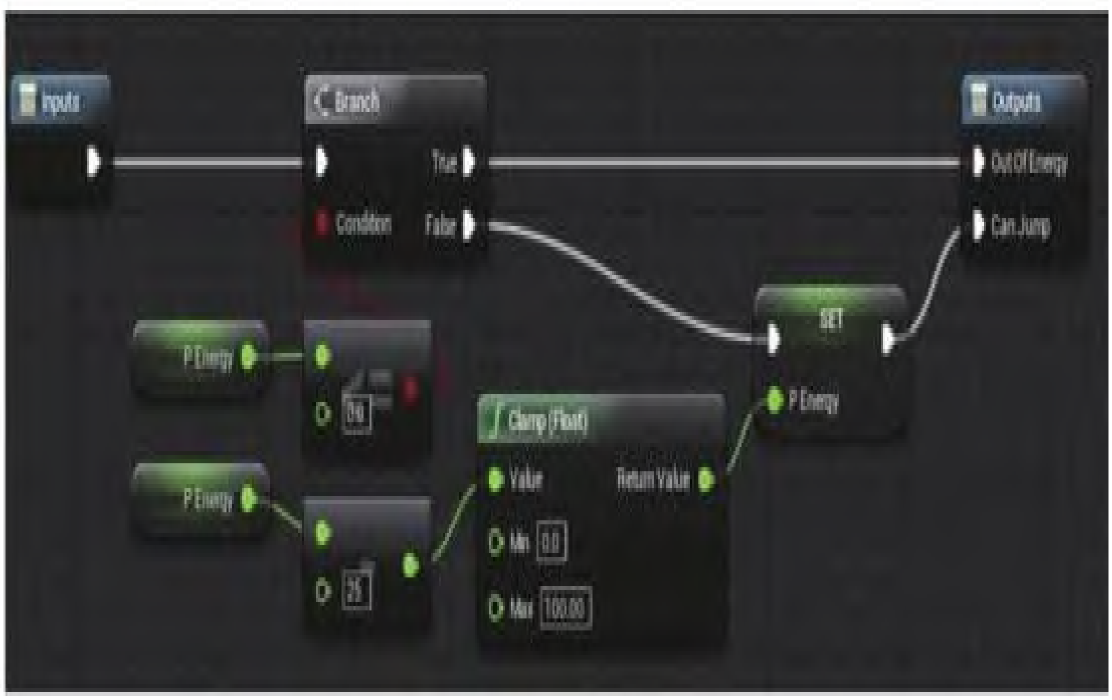



图3-142

利用此方法可随意添加输入或输出引脚。

下图事件图表应用了带额外引脚的宏，如图3-143所示。

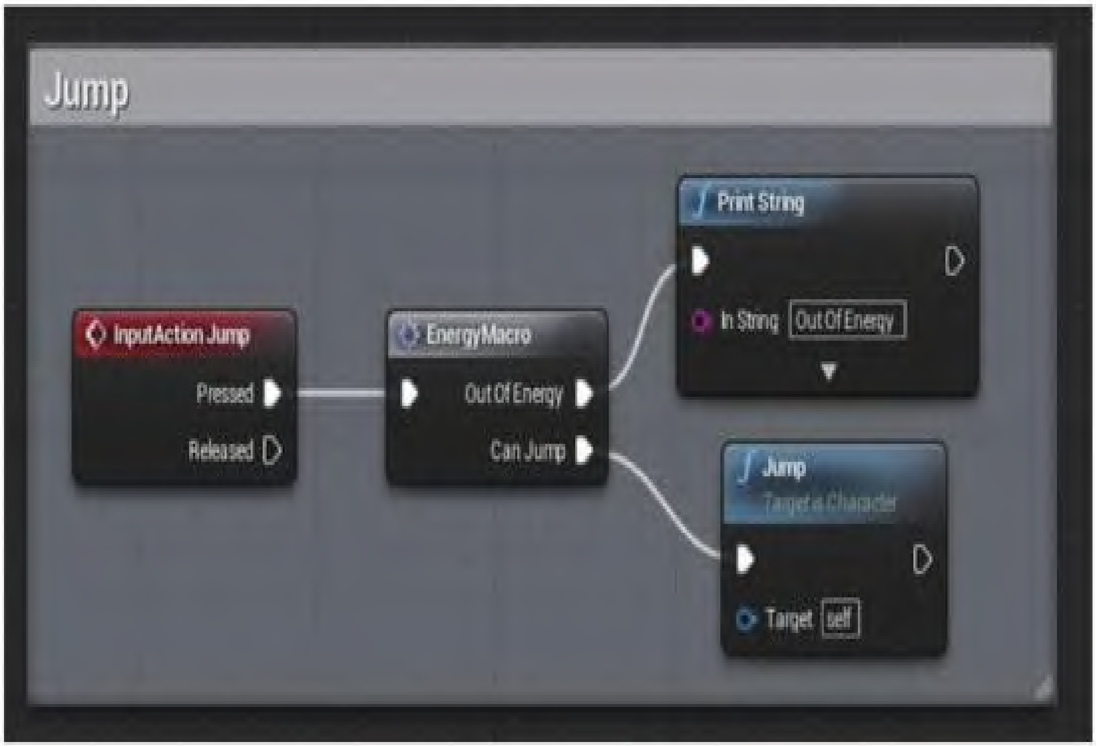



图3-143

Step 4　在宏上右击，可选择Expand Node将节点转换为其原始状态。

### 3.3.4　设置和获取Actor引用

通过建立对Actor的引用，你可以在玩游戏期间通过蓝图或C++访问其属性、变量、事件或函数（若适用），然后修改它们以满足你的游戏性需要。

例如，你想在玩家进入触发器盒时开灯，则建立对灯的引用，然后触发器盒将允许你通过将灯Actor键入触发器盒的OnComponentBeginOverlap事件来实现此目标（这样，你就可以开灯）。

正确引用Actor对Actor之间的通信也很重要，因为错误指定引用将导致失败的通信和不良结果。

可以查看以下步骤，了解设置和获取Actor引用的实例。

在此实例中，我们使用的是启用了新手内容的Blueprint Third Person模板。

**Step 1**　在Modes菜单中的Basic下，将一个Box Trigger拖入关卡，如图3-144所示。

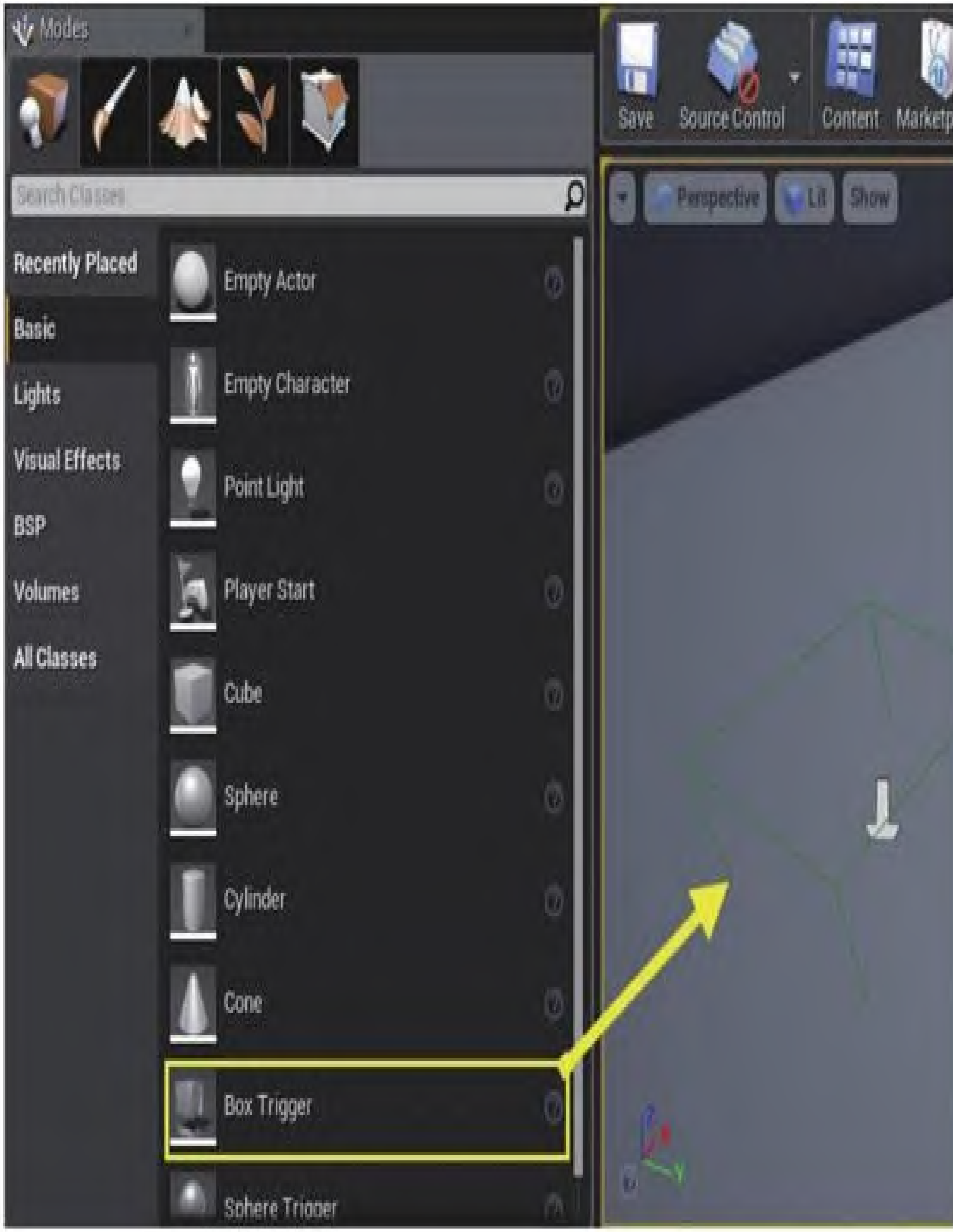



图3-144

我们将利用它触发关卡中另一个Actor的移动。

**Step 2**　在Box Trigger的Details面板中，选择Convert to Class Blueprint选项，如图3-145所示。

我们将使用这个Actor创建一个蓝图，并在其中指派一个存储引用Actor的变量。

**Step 3**　为蓝图命名，然后选择Create Blueprint（蓝图编辑器将打开）。

**Step 4**　在蓝图编辑器中的My Blueprint下，单击Add Variable按钮，如图3-146所示。

**Step 5**　单击变量，然后在Details面板中单击Variable Type按钮并在Object Reference下选择Actor，如图3-147所示。

在此我们指定引用为Actor类型，且为关卡中的一个Object。

**Step 6**　将变量命名为“TargetActor”并单击Editable复选框，如图3-148所示。

图3-145

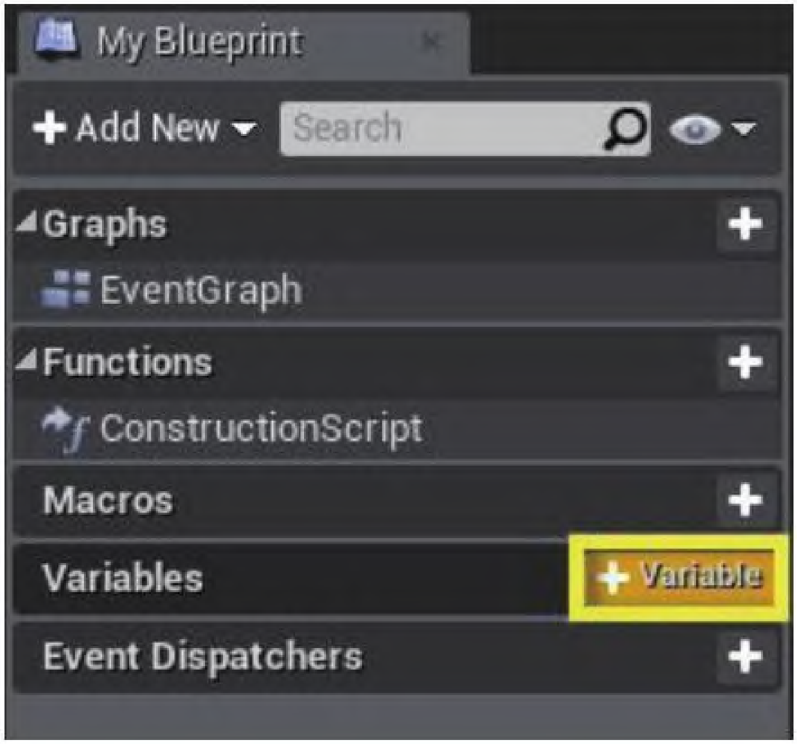

My Blueprint
Add New
Search
Graphs
EventGraph
Functions
ConstructionScript
Macros
Variables
Variable
Event Dispatchers

图3-146

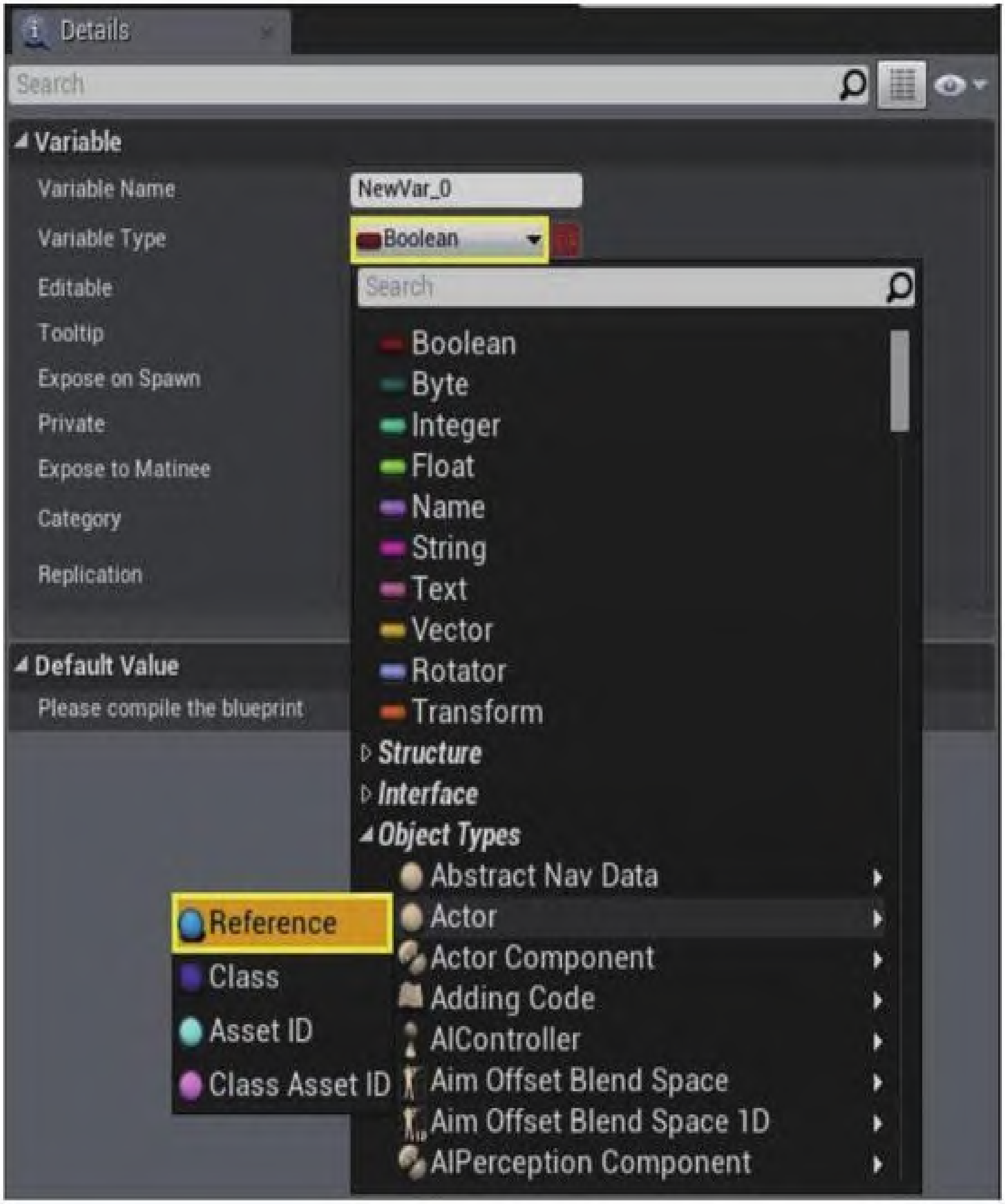

Details
Search
Variable
Variable Name
NewVar_0
Variable Type
Boolean
Editable
Tooltip
Expose on Spawn
Private
Expose to Matinee
Category
Replication
Default Value
Please compile the blueprint
Search
Boolean
Byte
Integer
Float
Name
String
Text
Vector
Rotator
Transform
Structure
Interface
Object Types
Abstract Nav Data
Actor
Actor Component
Adding Code
AIController
Aim Offset Blend Space
Aim Offset Blend Space 1D
AIPerception Component
Reference
Class
Asset ID
Class Asset ID

图3-147

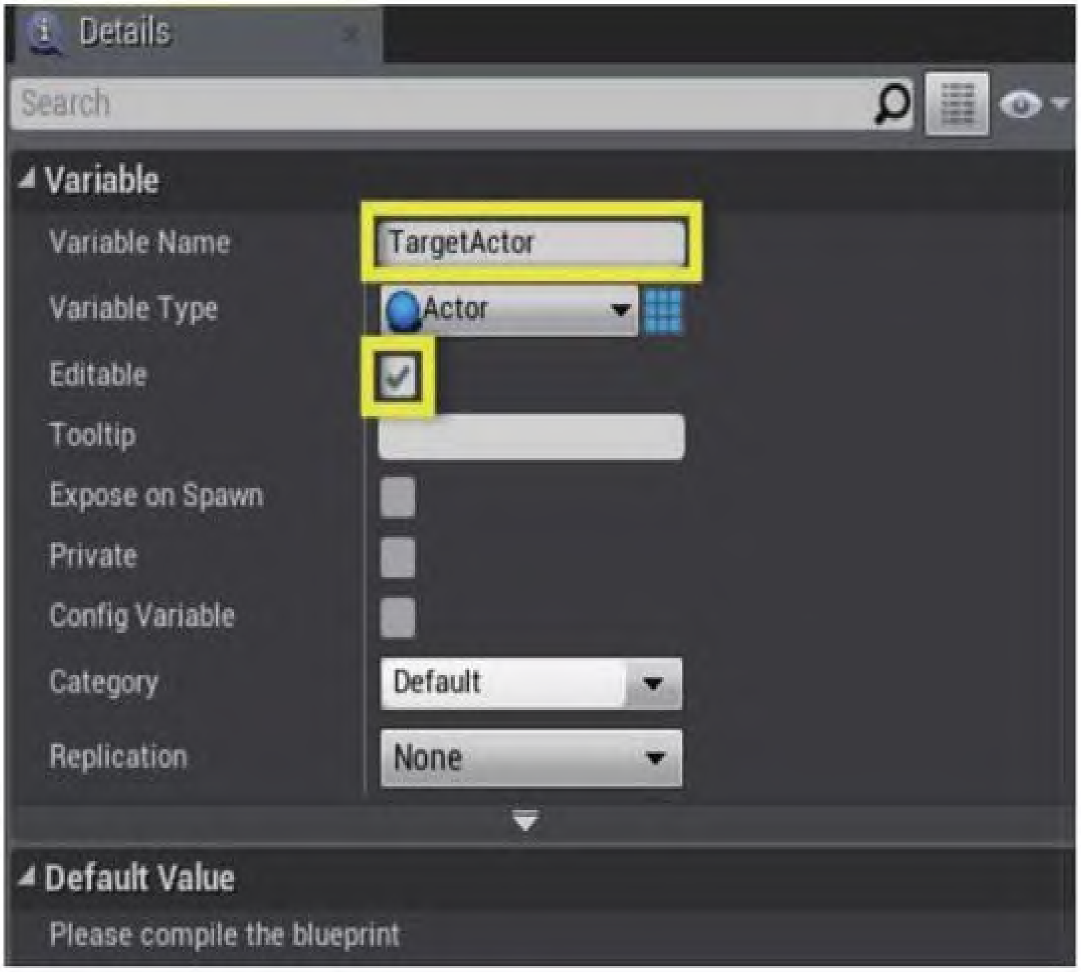



图3-148

此操作将变量设为Public，通过主编辑器窗口的Details面板可在此蓝图外对其进行访问和设置。该功能十分实用，无须进入蓝图即可对变量进行修改。

**Step 7**　可输入显示文本，作为变量功能和用途的提示文本，如图3-149所示。

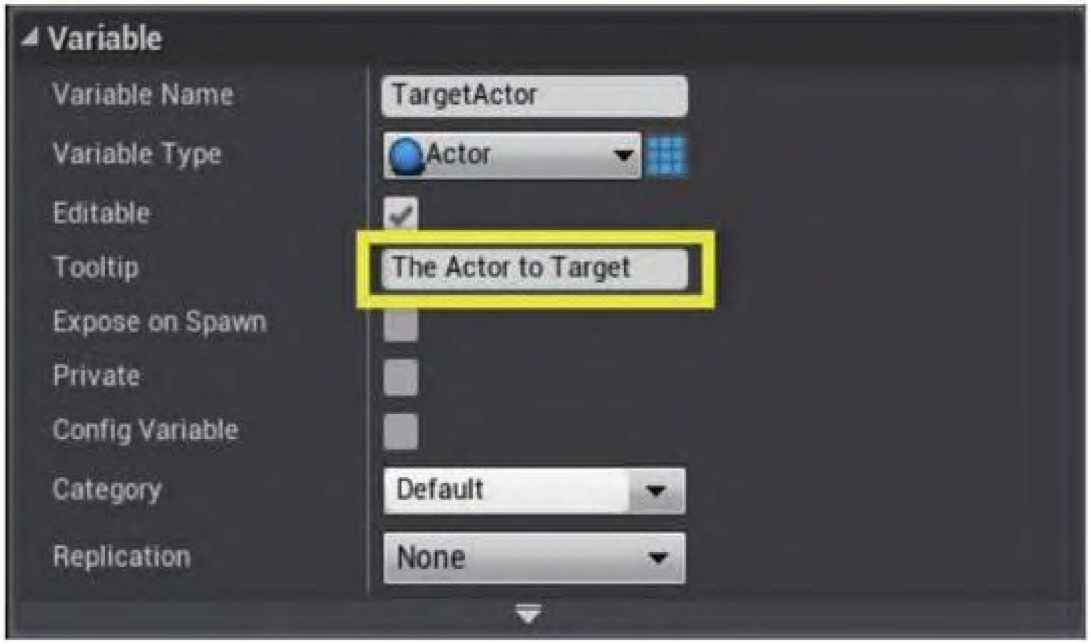



图3-149

在上图中输入的文本是“The Actor to Target”。

**Step 8** 单击控件蓝图编辑器窗口左上部的Compile按钮，如图3-150所示。

图3-150

完成编译后它将变为一个绿色勾选标记，如图3-151所示。

图3-151

Step 9　最小化蓝图，返回主编辑器窗口，然后在关卡中选择Box Trigger蓝图。

Step 10　在窗口的Details面板中，可以在Default下方看到Target Actor变量和提示文本，如图3-152所示。

Step 11　前往Content Browser的Content/StarterContent/Shapes路径，添加三个Shape_Cubes至关卡中，如图3-153所示。

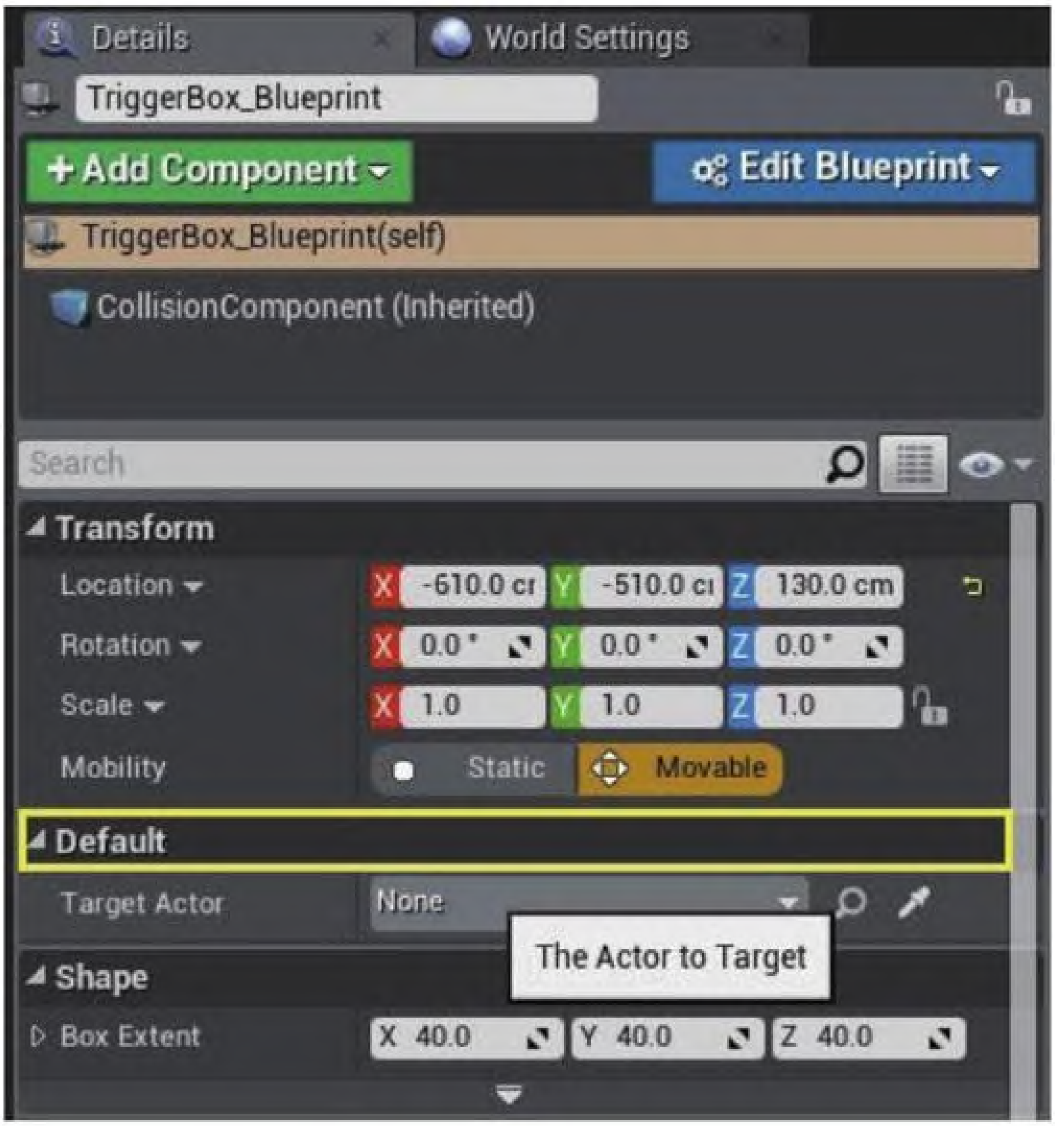



图3-152

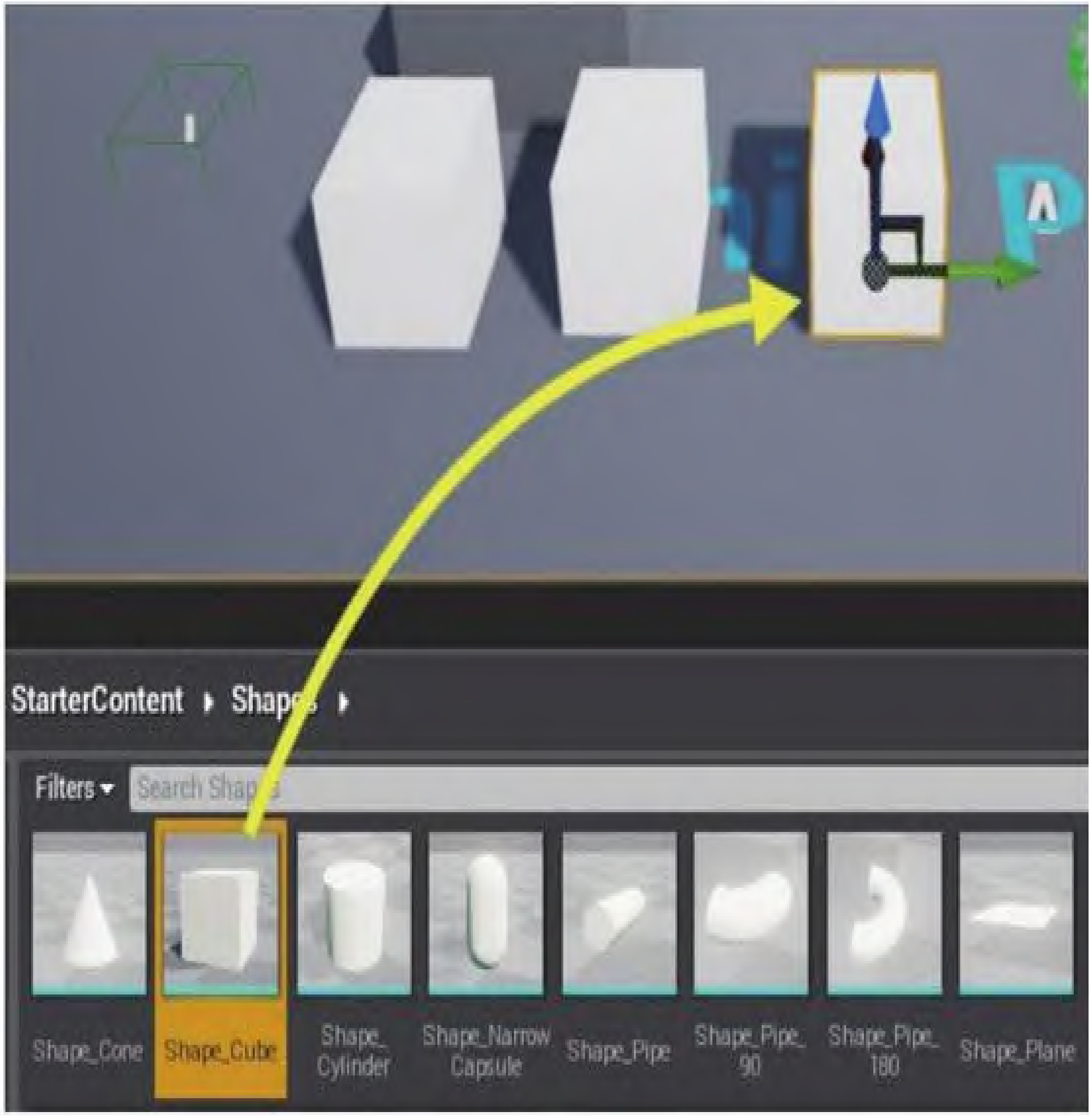



图3-153

如果使用的是自建项目，则可添加任意Actor、静止网格物体或蓝图到关卡中。

**Step 12** 向关卡添加资源后，可在每个Actor的Details面板中将其移动性设为Moveable，如图3-154所示。

完成此设置后，可通过蓝图脚本移动Actor。

**Step 13** 单击关卡中的Box Trigger蓝图。

**Step 14** 在Details面板中，单击Target Actor下的None框，搜索Shape[或为Actor（s）添加的其他名字]，如图3-155所示。

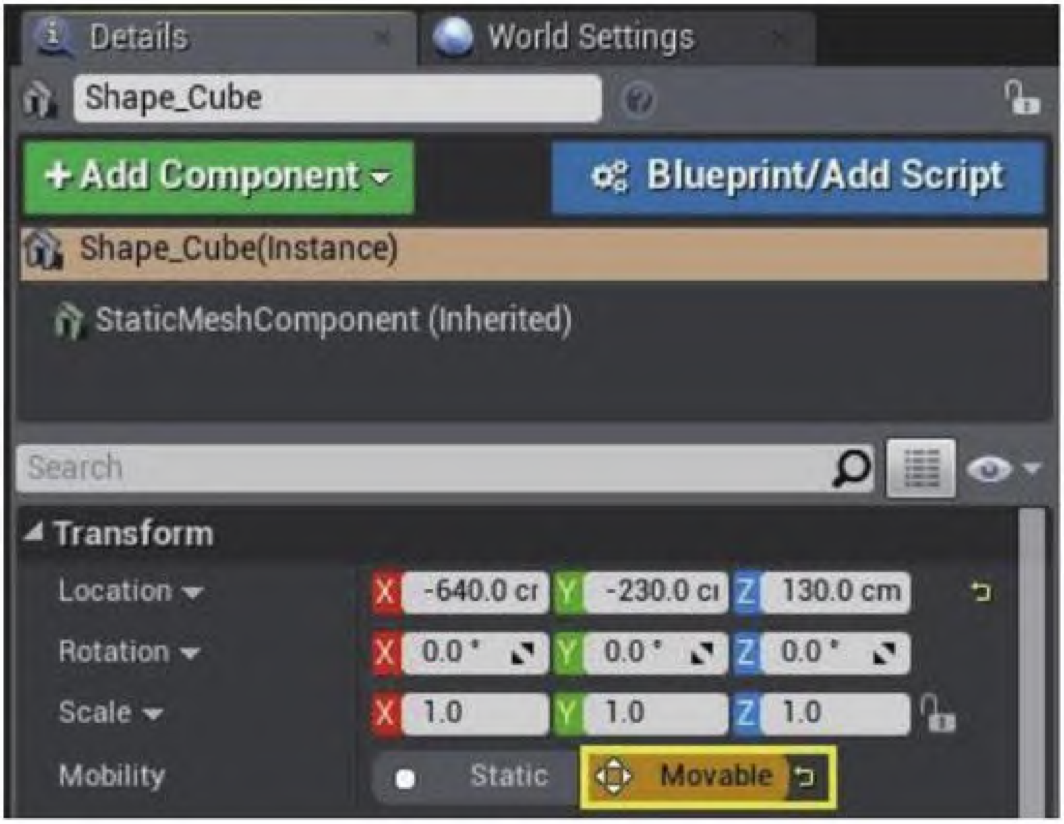

Details
World Settings
Shape_Cube
+ Add Component
Blueprint/Add Script
Shape_Cube(Instance)
StaticMeshComponent (Inherited)
Search
Transform
Location
-640.0 cr
-230.0 cı
130.0 cm
Rotation
0.0 °
0.0 °
0.0 °
Scale
1.0
1.0
1.0
Mobility
Static
Movable

图3-154

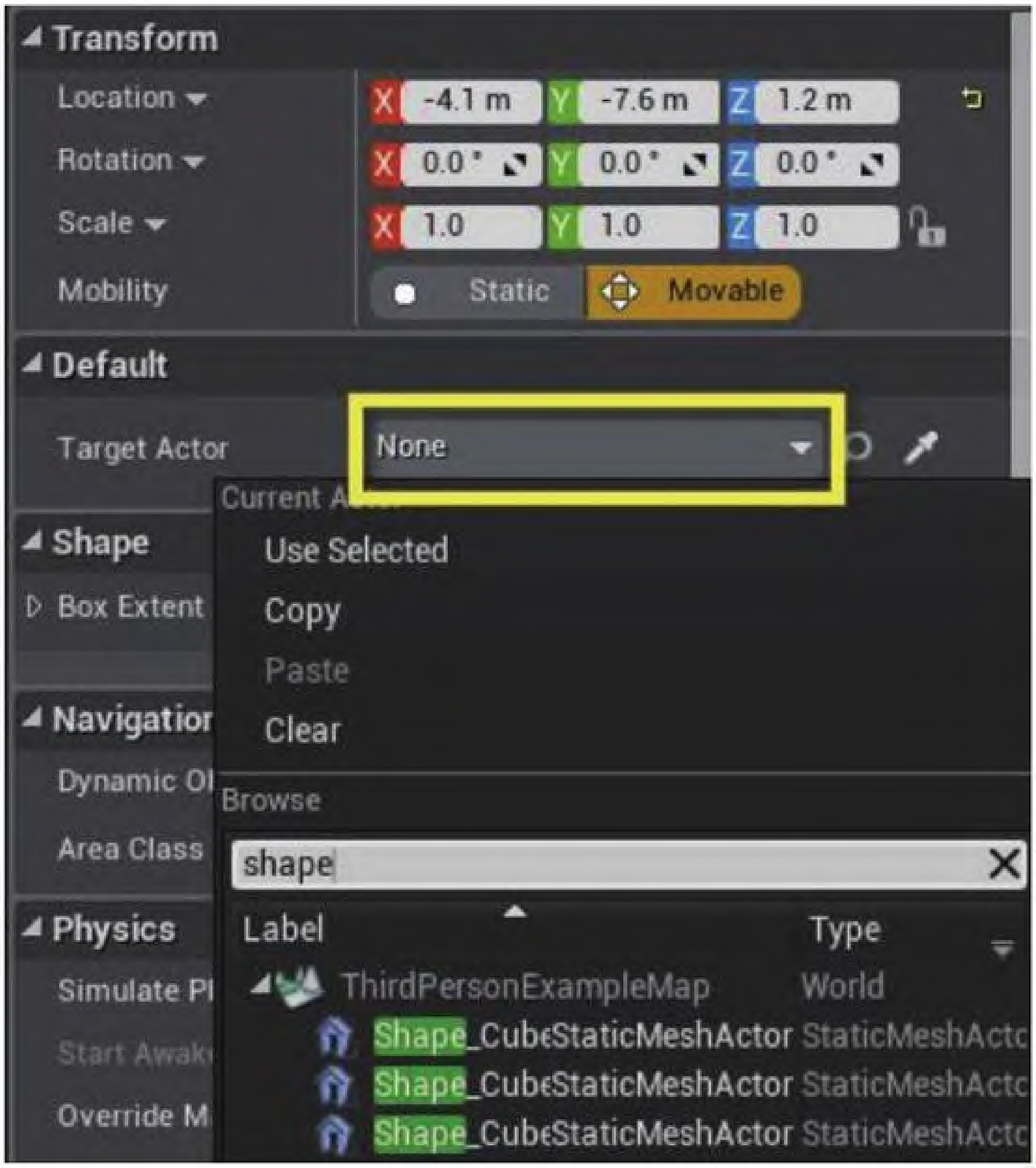



图3-155

单击None将显示当前关卡中的所有Actors，但是我们只需要将添加的方块之一作为目标。

**Step 15** 单击放大镜图标旁边的滴管图标，如图3-156所示。

在视口中单击它，将从关卡中选择一个Actor作为目标Actor。

**Step 16** 将滴管图标移动至一个方块（或添加的其他资源）上，单击将其设为目标Actor，如图3-157所示。

Transform
Location
X -4.1 m
Y -7.6 m
Z 1.2 m
Rotation
X 0.0°
Y 0.0°
Z 0.0°
Scale
X 1.0
Y 1.0
Z 1.0
Mobility
Static
Movable
Default
Target Actor
None
Pick Actor from scene
Shape
Box Extent
X 40.0
Y 40.0
Z 40.0


图3-156

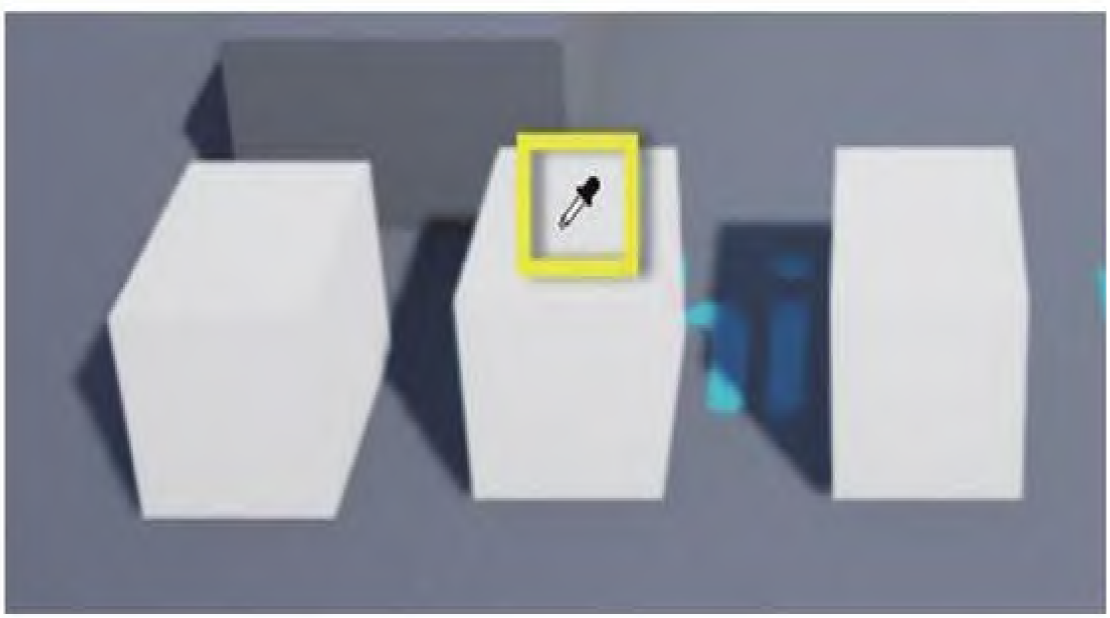


图3-157

**Step 17**　选择Actor时，Target Actor变量将更新为Details面板中所选的Actor，如图3-158所示。

成为目标Actor的Actor将拥有Set引用。

**Step 18**　打开Box Trigger蓝图，然后在MyBlueprint窗口中单击眼睛图标并启用ShowInherited Variables，如图3-159所示。

图3-158

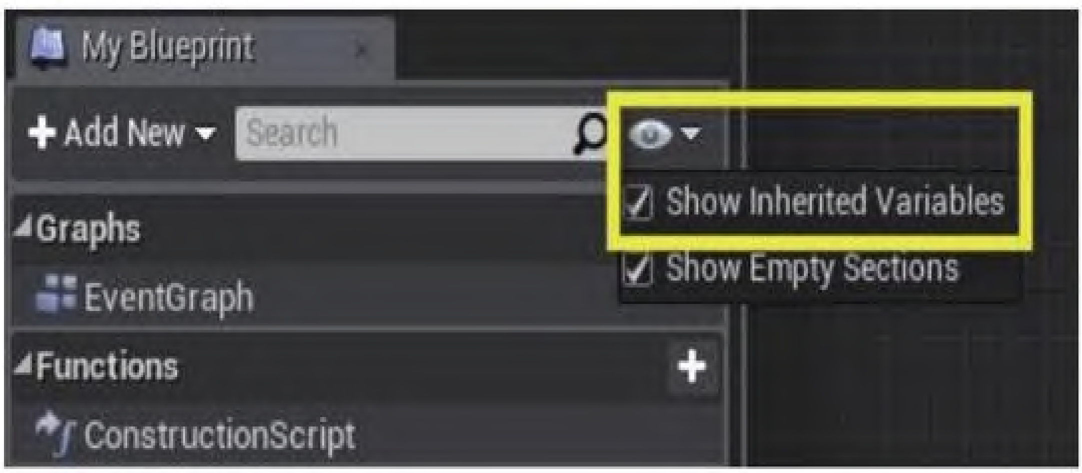



图3-159

此操作将在组件上添加事件。这些组件在最初将触发框转换为类蓝图时被继承。

**Step 19**　在Trigger Base下，在CollisionComponent上右击，选择Add Event，然后再选择Add OnComponentBeginOverlap，如图3-160所示。

图3-160

此操作将在图表上创建一个新的Event节点。

**Step 20** 按住Ctrl键将TargetActor变量拖入图表，如图3-161所示。

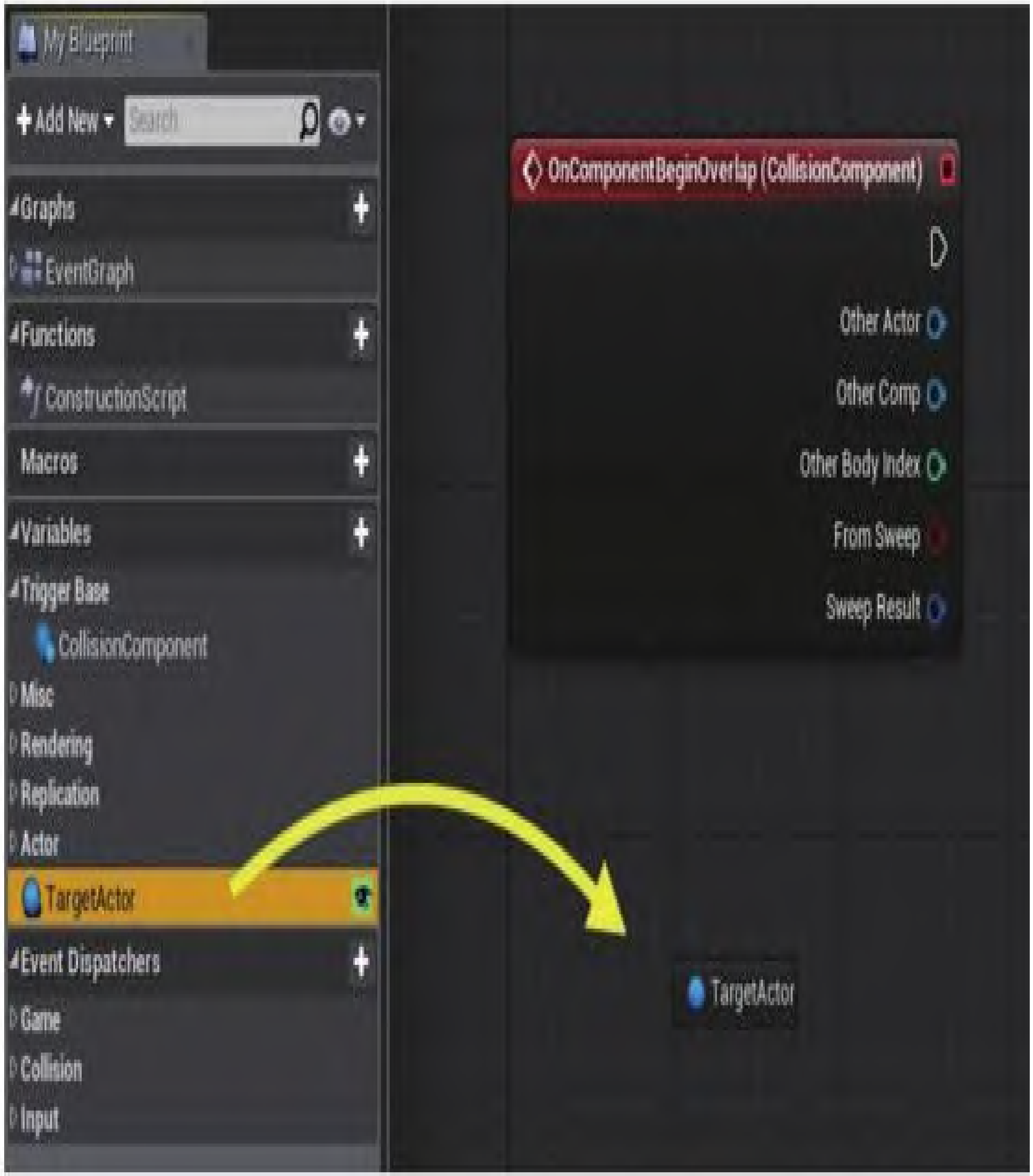



图3-161

此操作将创建一个Get节点。此节点将获取指派给变量的所有内容，而该变量已被设为关卡中的一个Actor。

**Step 21** 拖动TargetActor节点，搜索并添加一个Add Actor Local Offset节点，如图3-162所示。

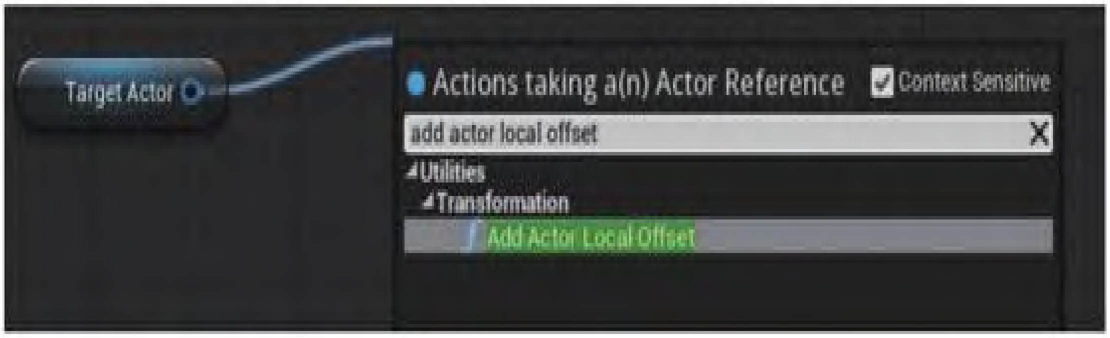



图3-162

此操作将添加当前目标的变化（可将其从当前位置移动）。

**Step 22** 以下图所示方式连接节点，然后将Add Actor Local Offset节点上的Z轴数值设为500.0，如图3-163所示。

图3-163

此操作将获取目标的当前位置，并在触发框发生重叠时使目标沿Z轴上移500个单位。

**Step 23** 在Components窗口中，单击选中TriggerBox_Blueprint，如图3-164所示。

如果Components窗口不存在，可使用Window菜单选项将其启用，如图3-164A、图3-164B所示。

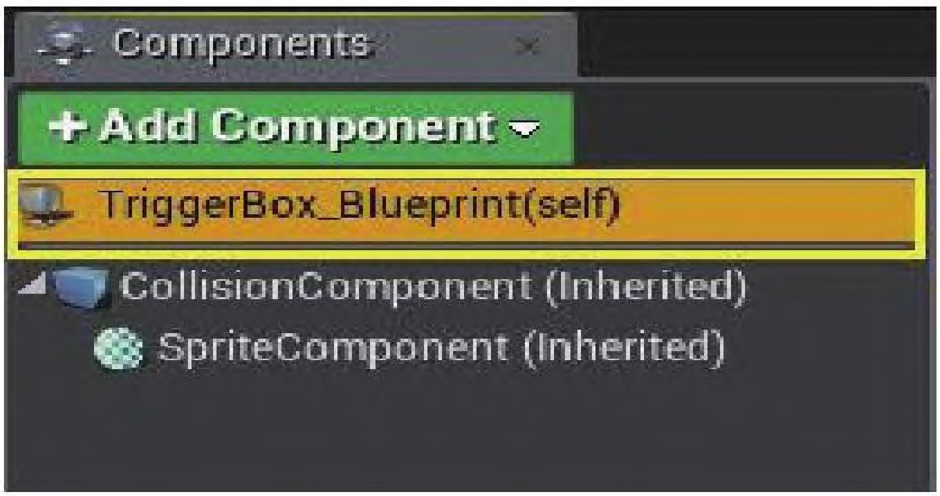



图3-164A

图3-164B

**Step 24** 在触发框Details面板的Rendering下，取消勾选Actor Hidden In Game选项，如图3-165所示。

此操作将使触发框在游戏中可见，便于判定玩家从何处进入。

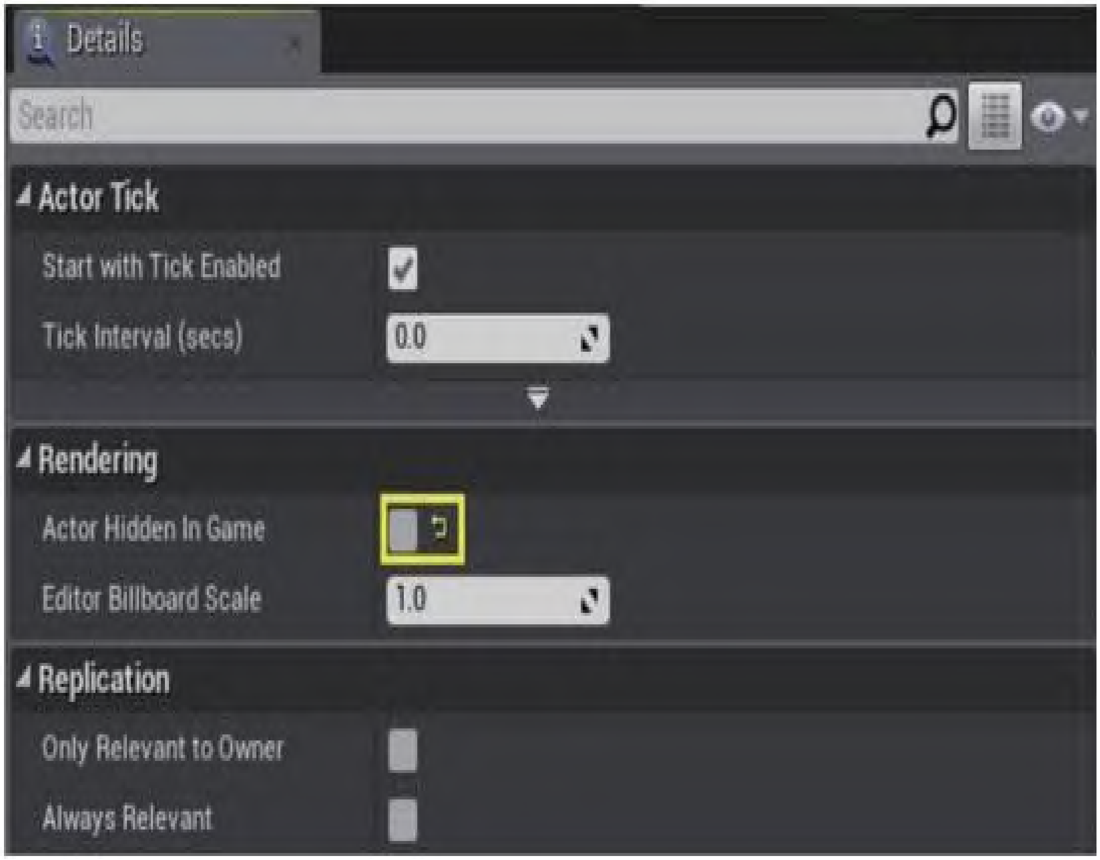
**Step 25** 单击Compile和Save，然后关闭蓝图。

**Step 26** 在主编辑器窗口中，单击Play图标在编辑器中开始游戏。

当你每次进入触发框时，选定的Actor均将上移500个单位。建立引用后，变量便了解应该在触发框脚本中影响哪个Actor。尝试利用该脚本影响关卡中的另一个Actor。

**Step 1** 在关卡视口的主编辑器窗口中单击Box Trigger蓝图。

**Step 2** 在Box Trigger的Details面板中，单击Target Actor旁的滴管图标，如图3-166所示。



Details
Search
Actor Tick
Start with Tick Enabled
Tick Interval (secs)
0.0
Rendering
Actor Hidden In Game
Editor Billboard Scale
1.0
Replication
Only Relevant to Owner
Always Relevant


图3-165

Transform
Location
X -4.1 m
Y -7.6 m
Z 1.2 m
Rotation
X 0.0°
Y 0.0°
Z 0.0°
Scale
X 1.0
Y 1.0
Z 1.0
Mobility
Static
Movable
Default
Target Actor
None
Pick Actor from scene
Shape
Box Extent
X 40.0
Y 40.0
Z 40.0

图3-166

**Step 3**　在关卡视口中单击一个从未使用的Actor。

**Step 4**　再次在编辑器中开始游戏。

在此情况下，每次玩家进入触发框都会使所选的新Actor沿Z轴上移500个单位。通过对触发框脚本中Target Actor变量的使用，与脚本相关的特定Actor便不会被锁定并可对其进行修改；此外无须对蓝图进行修改便可改变其影响的Actor。

该示例说明如何将不同对象四处移动。除移动对象外，可使用该方法在玩家按键时开门、开关不同光源、生成供玩家拾取或销毁的道具，以及其他各种功能。

这也是为Actor获取引用的多种方式之一。如需要为玩家角色获取引用却无法在主编辑器窗口清楚地进行设置，则须使用不同方法。

## 3.3.5　创建函数

Functions（函数）是属于特定蓝图的节点图表，该节点图表可以执行或者从蓝图中的另一个图表中进行调用。函数具有一个单一的入口点，由和该函数具有相同名称的一个节点指定，该节点包含了一个单一的执行输出引脚。当从另一个图表调用函数时，会激活该输出执行引脚，促使执行连接网络。

以下步骤将说明创建按键时可在屏幕上显示文本的函数。

**Step 1**　在Content Browser中，单击New按钮，然后选择Blueprint Class，如图3-167所示。

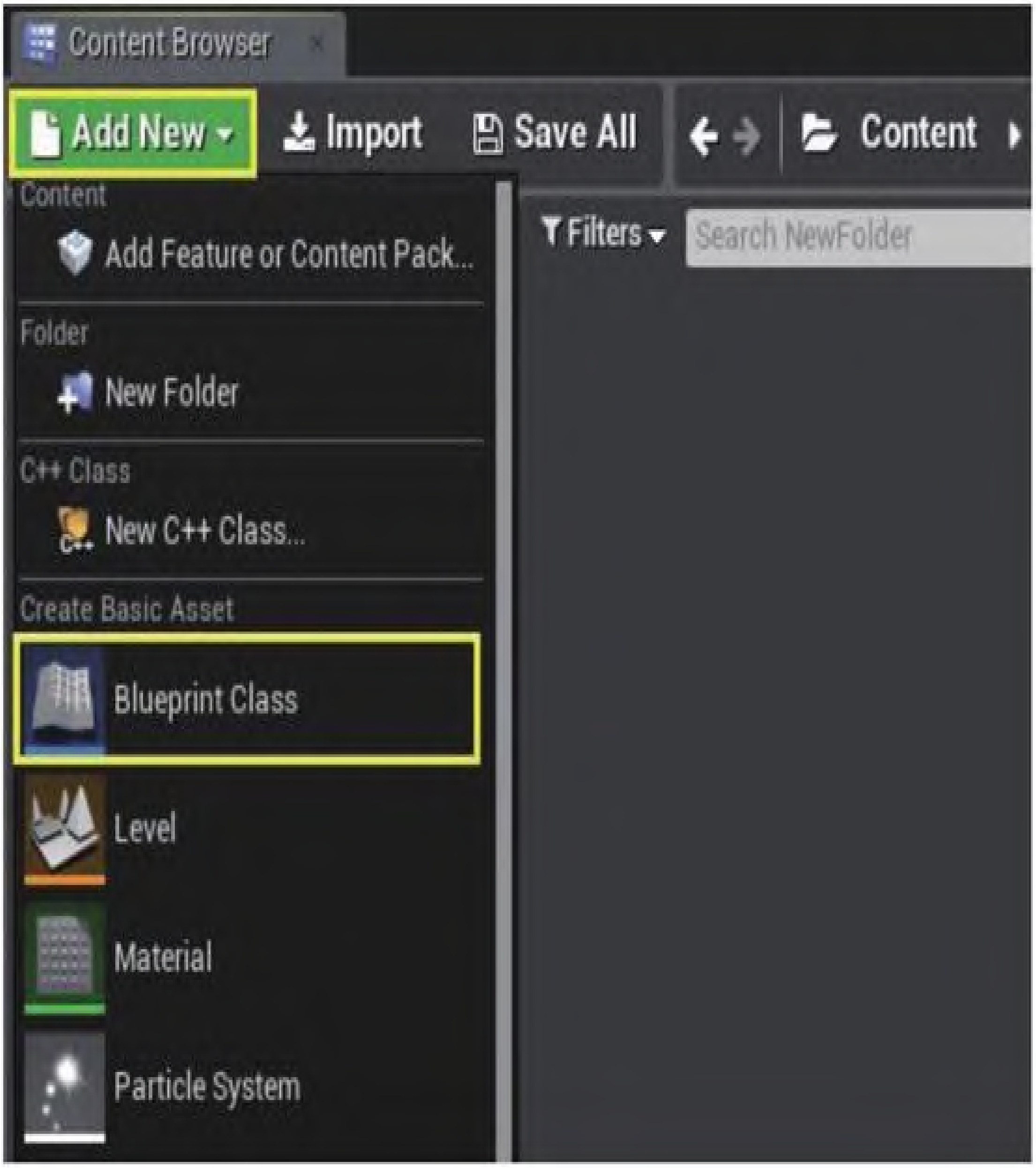



图3-167

**Step 2** 在Pick Parent Class窗口中选择Actor，如图3-168所示。

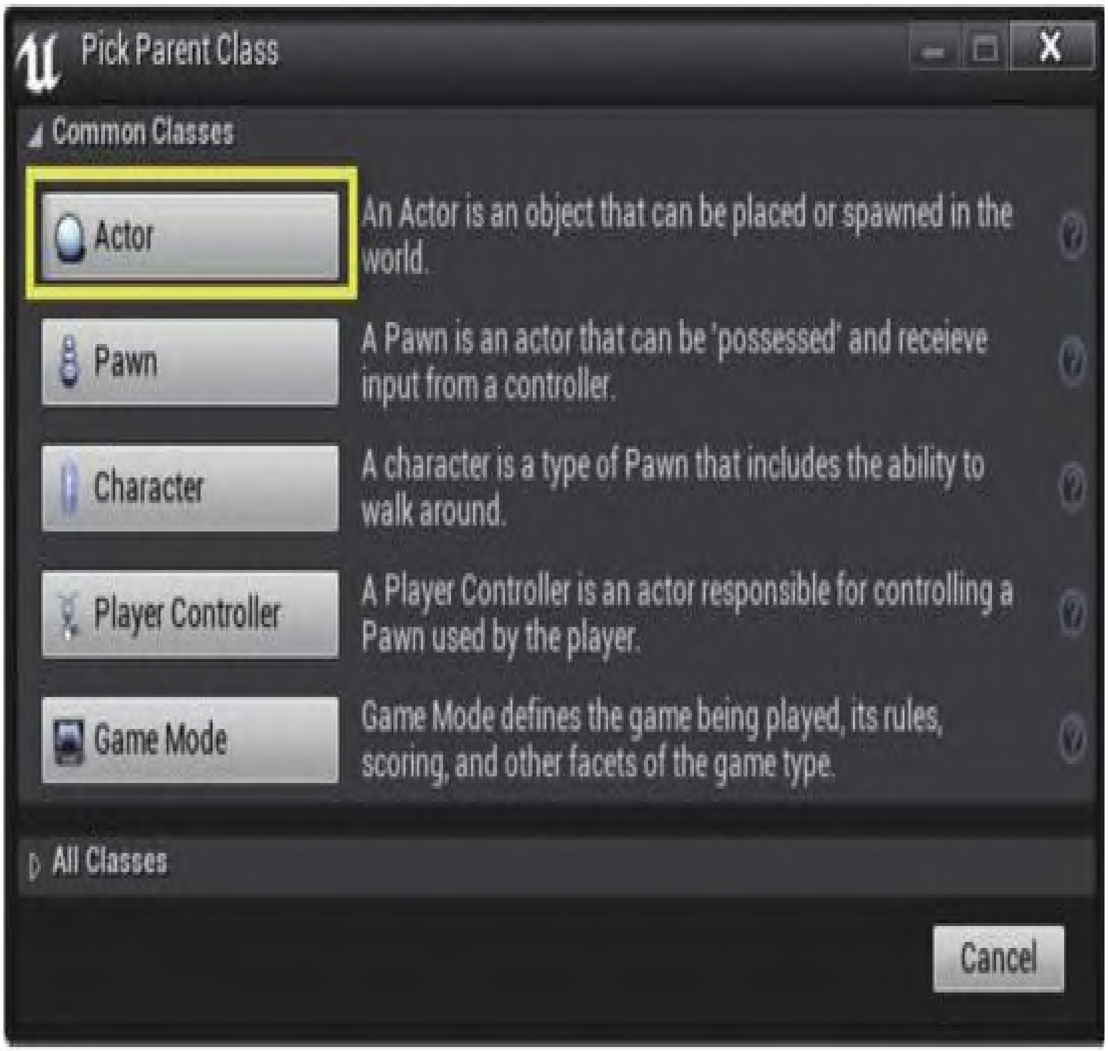



图3-168

**Step 3** 为蓝图命名，然后双击将其在蓝图编辑器中打开，如图3-169所示。

**Step 4** 在图表中右击，搜索并添加Event Begin Play事件，如图3-170所示。

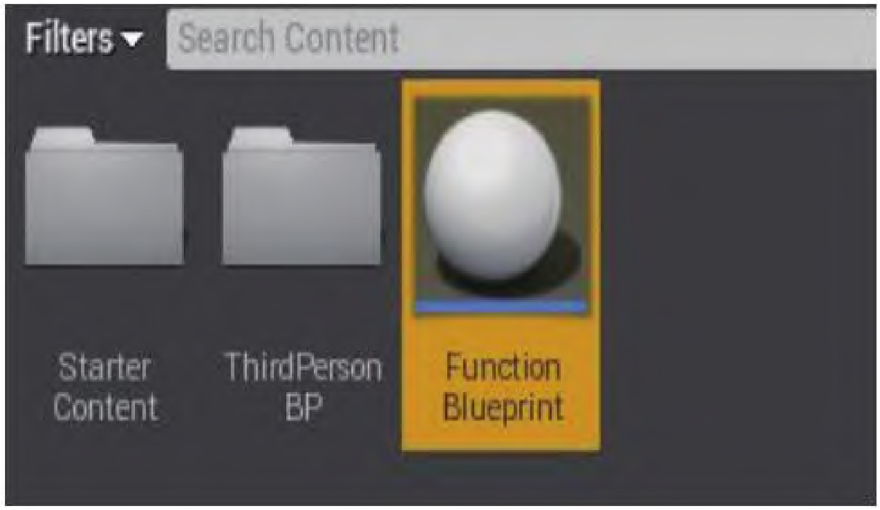



图3-169

图3-170

游戏启动后该节点便伴随其后的脚本开始执行。

**Step 5**　在图表中右击，搜索并添加Get Player Controller节点，如图3-171所示。

此操作将获取当前指派的玩家控制器，并为该蓝图启用输入。

**Step 6**　在图表中右击，搜索并添加Enable Input节点，如图3-172所示。

图3-171

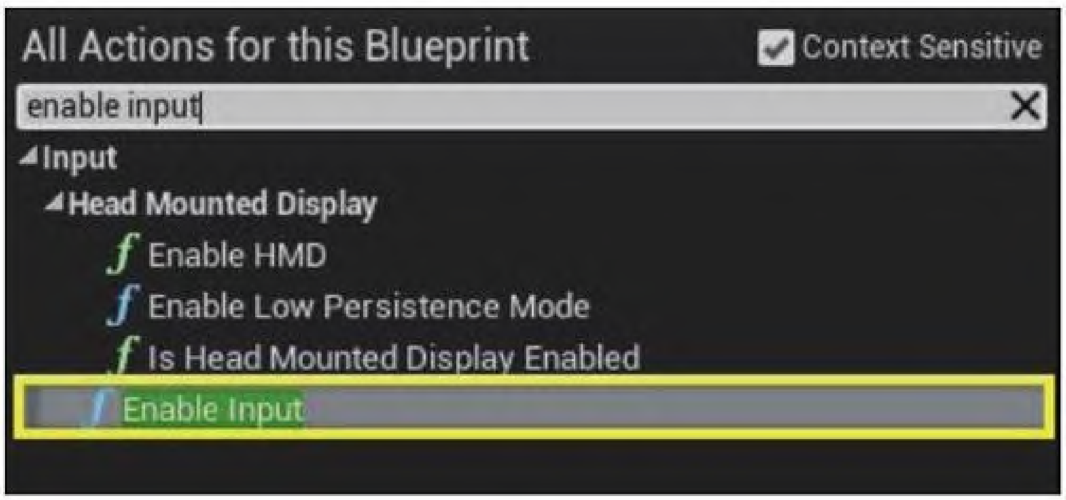



图3-172

此节点使输入被该蓝图接收。

**Step 7** 按下面的方法连接节点，如图3-173所示。

游戏启动后，选取玩家控制器并在该蓝图中启用控制器的输入。

**Step 8** 在MyBlueprint窗口中单击Add New Function按钮，如图3-174所示。

Event Begin Play
Enable Input
Target is Actor
Target
self
Player Controller
Get Player Controller
Player Index
0
Return Value

图3-173

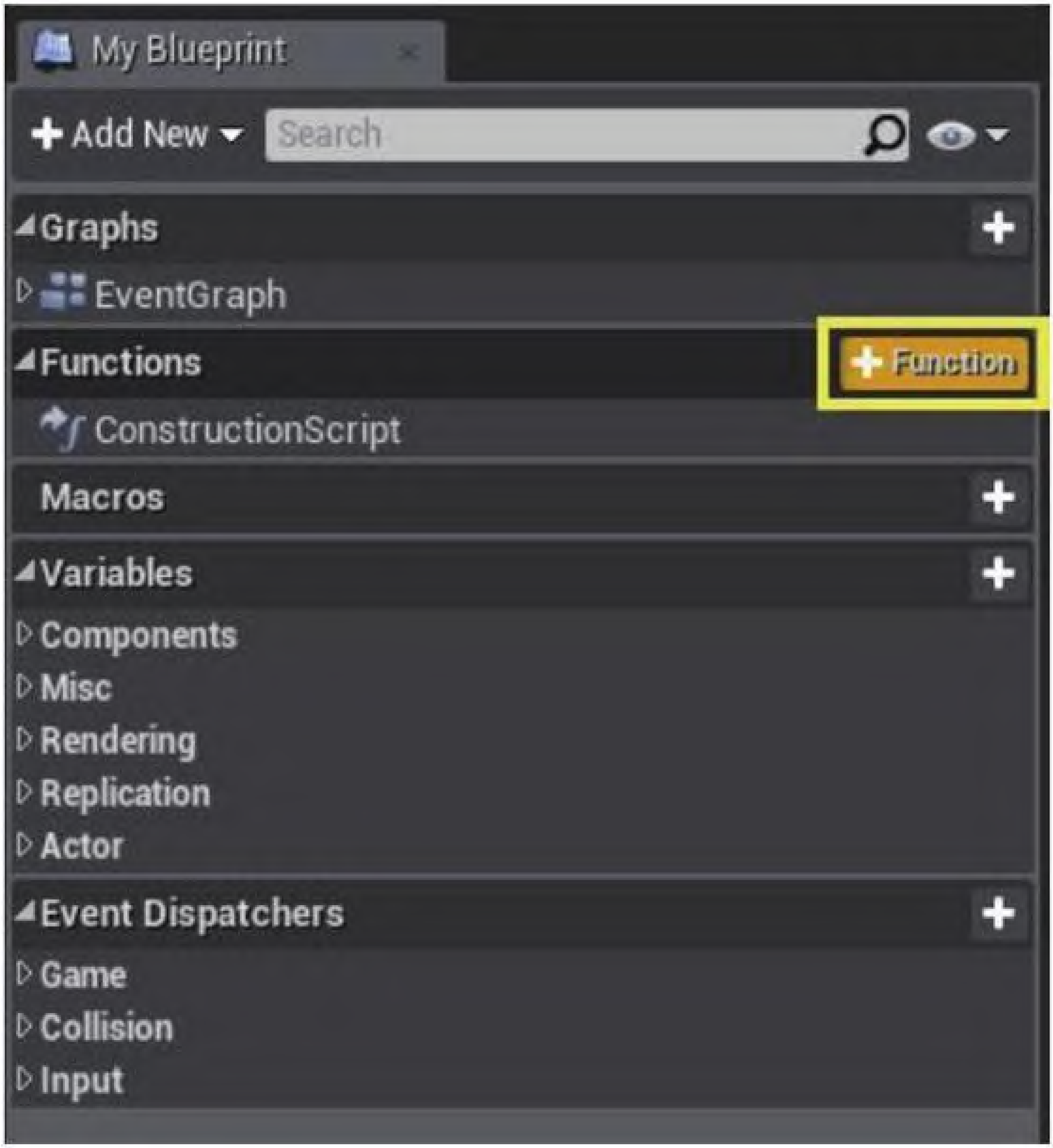



图3-174

**Step 9** 在My Blueprint窗口中选择新函数并按F2键对其重命名，如图3-175所示。

为函数命名，如“Print Text”。

**Step 10** 在函数图表中，拖动Print Text引脚，搜索并添加一个Print String节点，如图3-176所示。

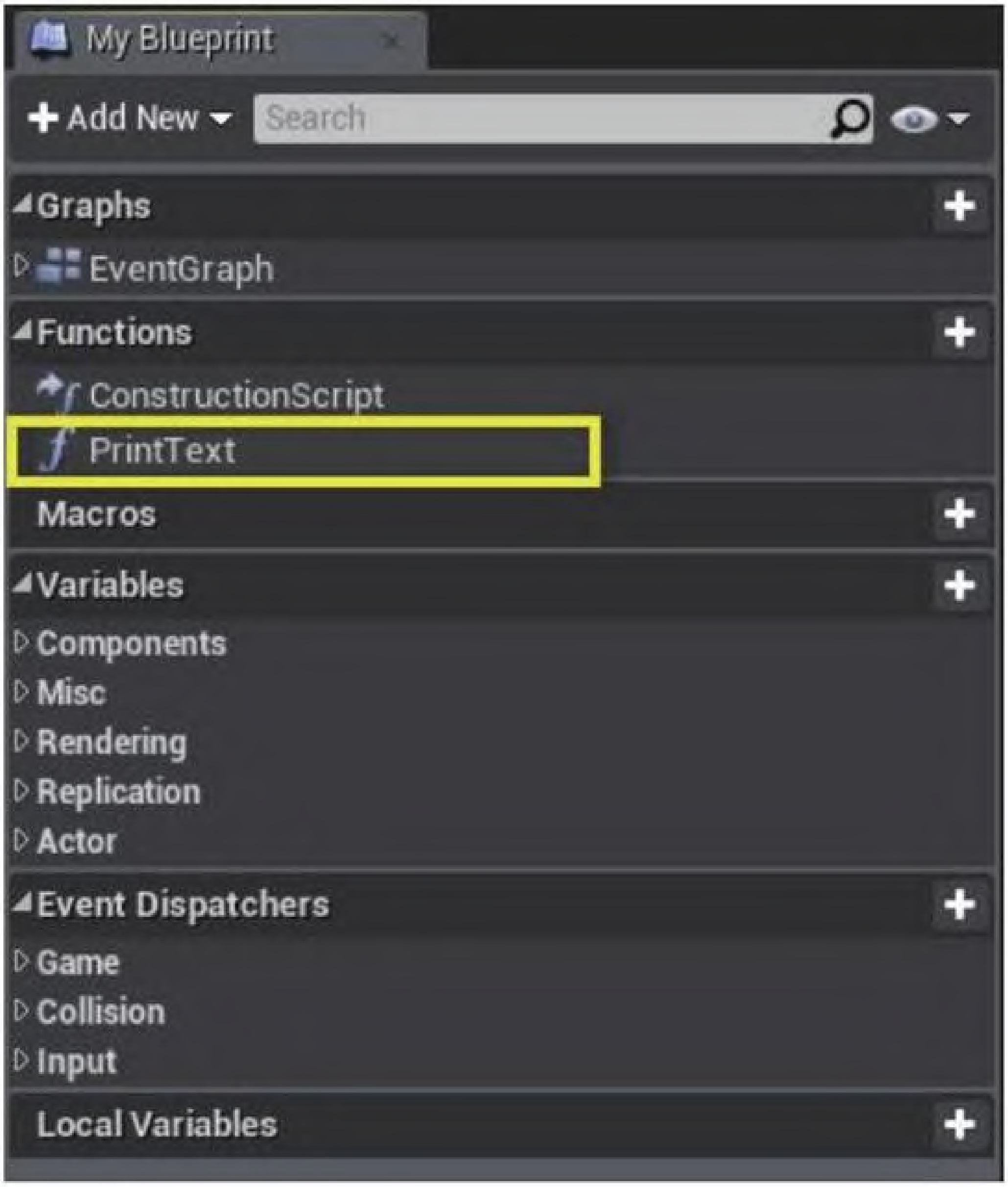

My Blueprint
Add New
Search
Graphs
EventGraph
Functions
ConstructionScript
PrintText
Macros
Variables
Components
Misc
Rendering
Replication
Actor
Event Dispatchers
Game
Collision
Input
Local Variables

图3-175

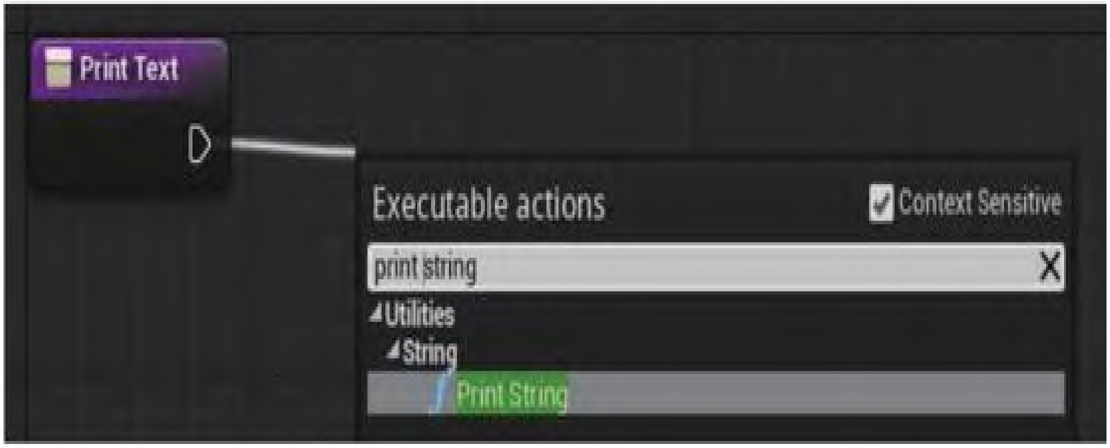



图3-176

**Step 11**　在In String框中，可对游戏中显示的文本进行修改，如图3-177所示。

图3-177

**Step 12**　单击Event Graph标签返回事件图表，如图3-178所示。

图3-178

**Step 13**　在图表中右击，搜索并添加一个F按键事件，如图3-179所示。

**Step 14** 拖动Pressed引脚，搜索并添加Print Text函数，如图3-180所示。

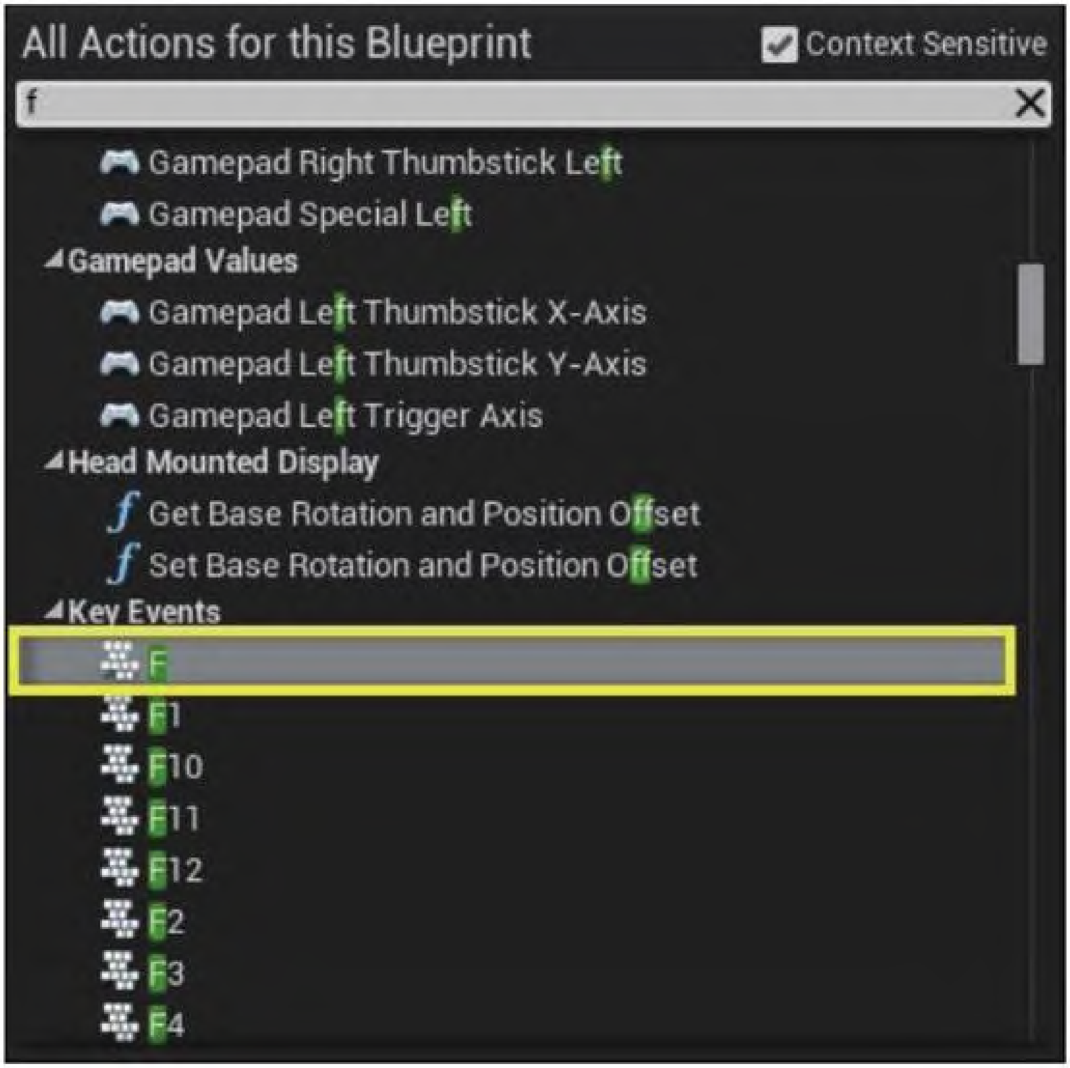



图3-179

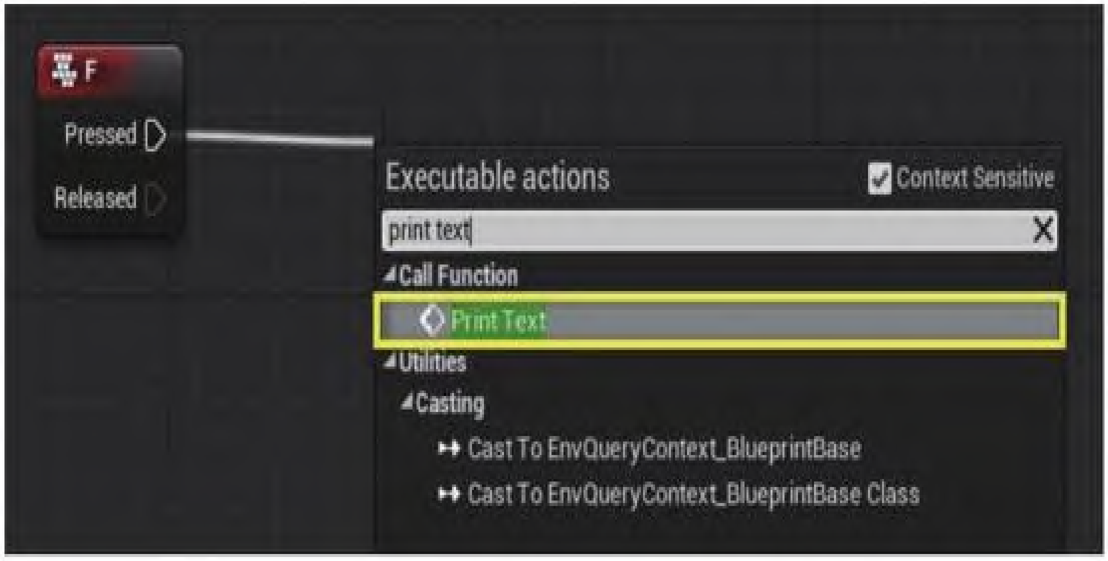



图3-180

按下F键时将调用Print Text函数，它使用Print String将文本显示到屏幕上。

如找不到函数，可单击工具栏中的Compile按钮，然后尝试重新搜索。

**Step 15**　网络外观与图3-181所示相似。

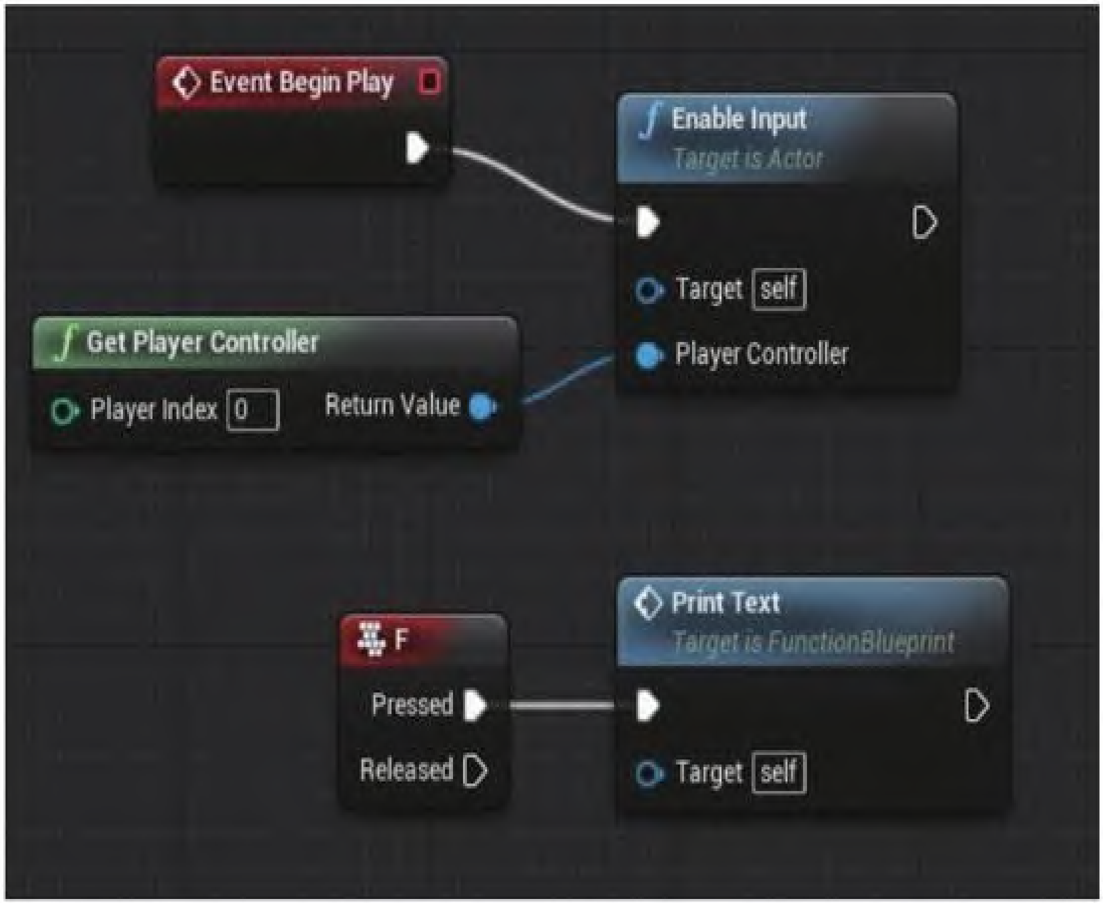



图3-181

**Step 16** 单击Compile按钮，然后关闭蓝图，如图3-182所示。

图3-182

**Step 17** 将蓝图拖入关卡，然后单击Play按钮在编辑器中开始游戏，如图3-183所示。

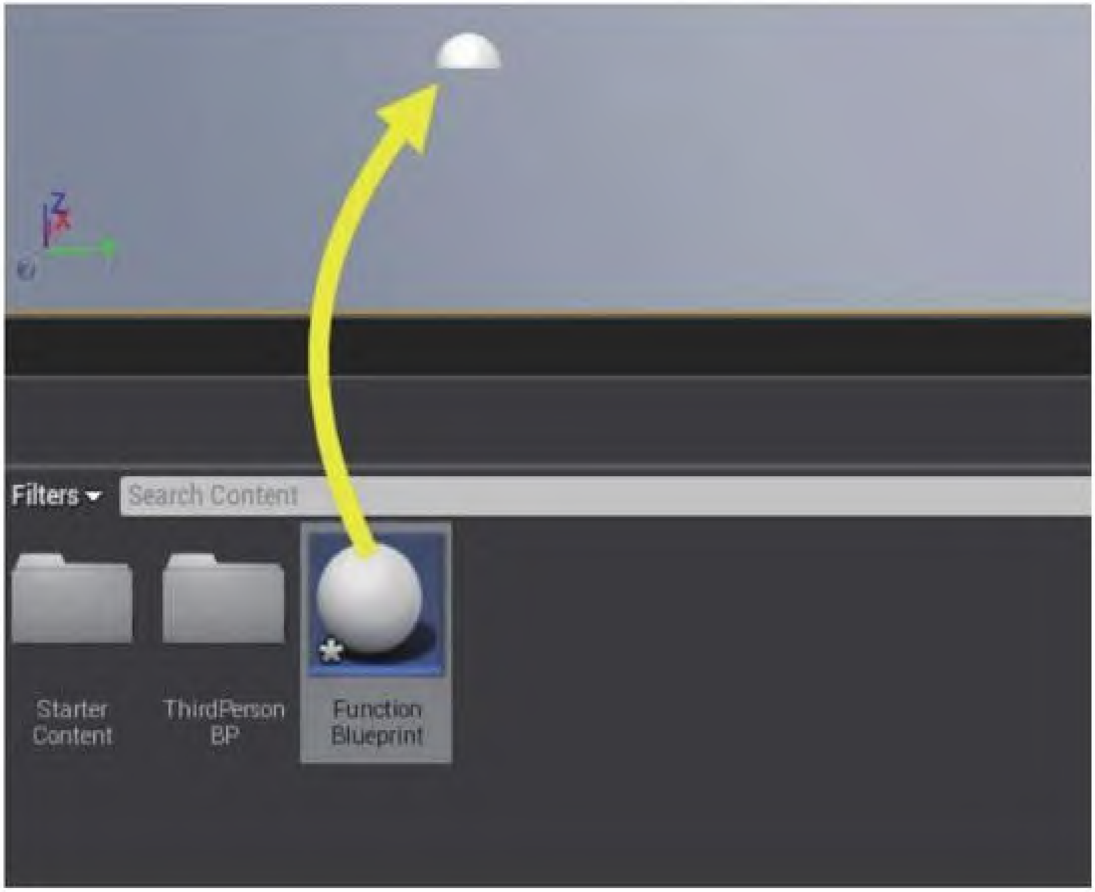



图3-183

**Step 18** 按下F键，函数将被调用，屏幕上将显示出文本，如图3-184所示。

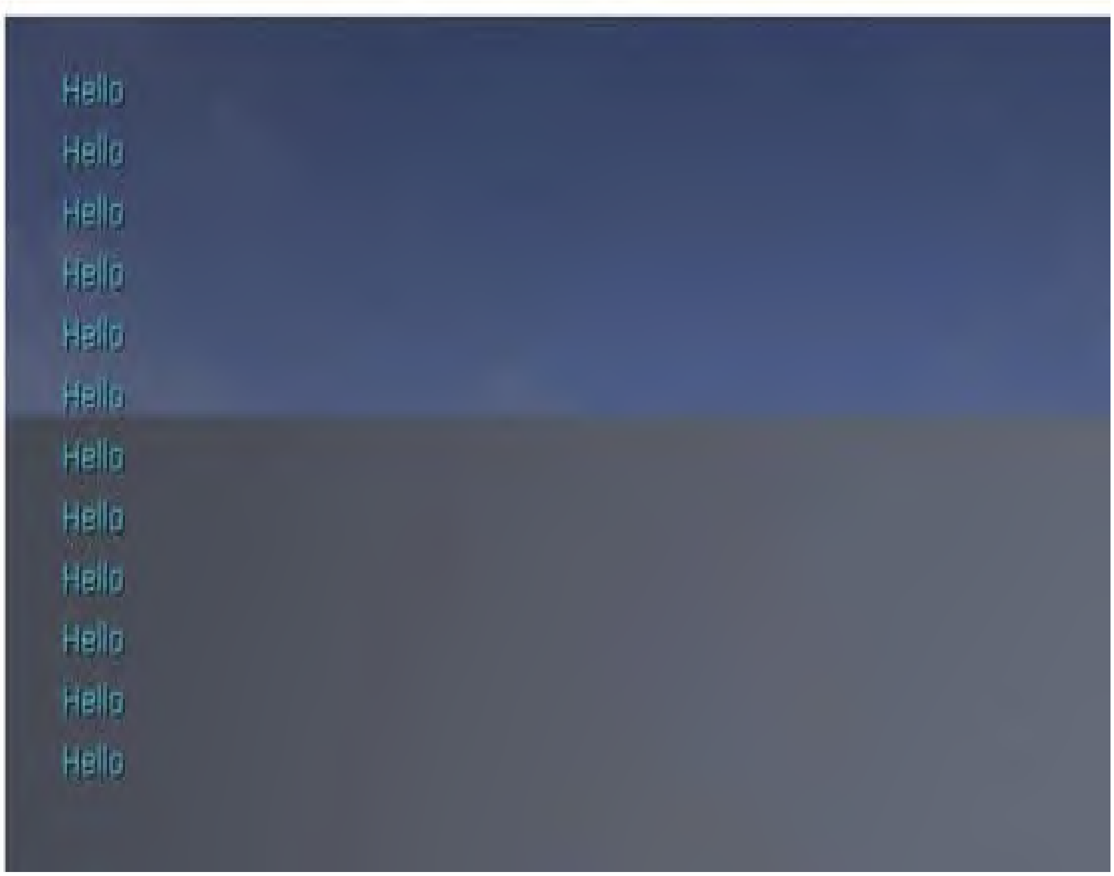



图3-184

虽然这个范例函数只能在按键时在屏幕上显示文本，但也可以为其添加更多指定键按下时执行的脚本。

举例而言，函数可用于按键发生时施出魔法，脚本则包含魔法的生成、位置、魔法的相关效果、对游戏世界场景的影响、是否对其他Actors造成伤害等内容。它们可全部包含在函数中，使事件图表中不包含函数中已有的脚本。

## 3.3.6　蓝图用户指南：变量

### 3.3.6.1　概述

Variables（变量）是存放一个值或引用世界中的一个Object或Actor的属性。这些变量可以让用户在界面内部进行访问，或者通过设置使得可以在外部进行访问，以便应用放置在关卡中的蓝图实例的设计人员可以修改它们的值。

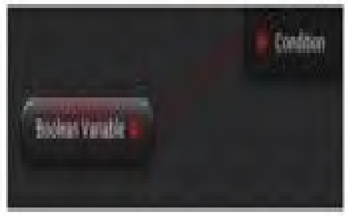
变量显示为圆角方框，方框内包含了变量的名称，如图3-185所示。

图3-185

关于使用Matinee修改特定类型的变量的信息，请参照使用Matinee修改蓝图属性的方法。

### 3.3.6.2　变量类型

你可以创建各种类型的变量，包括数据类型的变量（比如布尔型、整型及浮点型）及用于存放类似于Object、Actor及特定类的引用型变量。你还可以创建每种变量类型的数组。每种变量类型都进行了颜色编码，以方便识别，具体见下表。

| 变量类型 | 颜色 | 示例 | 应用 |
|---|---|---|---|
| Boolean（布尔型） | 红色 |  | 红色变量代表布尔型（true/false）数据 |
| Integer（整型） | 蓝绿色 | | 蓝绿色变量代表整型数据或者没有小数位的数值，比如 0、152 和 -226 |
| Float（浮点型） | 绿色 | | 绿色变量代表浮点型数据或具有小数位的数值，比如 0.0553、101.2887 和 -78.322 |
| String（字符串） | 洋红色 | | 洋红色变量代表字符串型数据或者一组字母数字字符，比如 Hello World |
| Text（文本） | 粉色 | | 粉色变量代表显示的文本数据，尤其是在文本需要进行本地化的地方 |

续表

| 变量类型 | 颜色 | 示例 | 应用 |
|---|---|---|---|
| Vector（向量） | 金黄色 | | 金黄色变量代表向量型数据，或者说是代表由三个浮点型数值的元素或坐标轴构成的数值，比如 XYZ 或 RGB 信息 |
| Rotator（旋转量） | 紫色 | | 紫色变量代表旋转量数据，这是一组在三维空间中定义了旋转度的数值 |
| Transform（变换） | 橙色 | | 橙色变量代表变换数据，它包括平移（三维位置）、旋转及缩放 |
| Object（对象） | 蓝色 | | 蓝色变量代表 Objects（对象），包括 Lights、Actors、StaticMeshes、Cameras 及 SoundCues |

### 3.3.6.3　我的蓝图选卡中的变量

我的蓝图面板允许给蓝图添加自定义的变量，且列出了所有的现有变量，包括在组件列表中添加的组件实例变量或通过在图表中通过把一个值提升为变量而创建的变量，如图3-186所示。

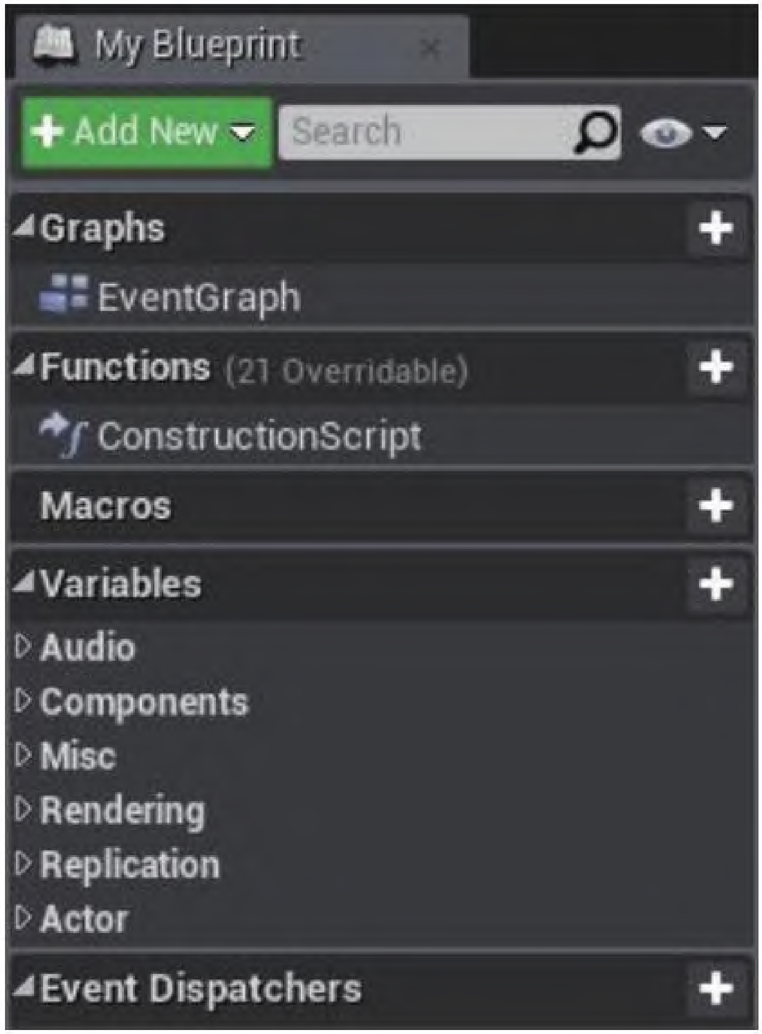
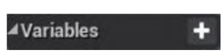



图3-186

#### 3.3.6.4 创建变量

在我的蓝图中，通过使用 

按钮可以直接添加新变量。你可以在我的蓝图选卡中命名该变量，并在详细信息选卡中设置其类型及其他属性，如图3-187所示。

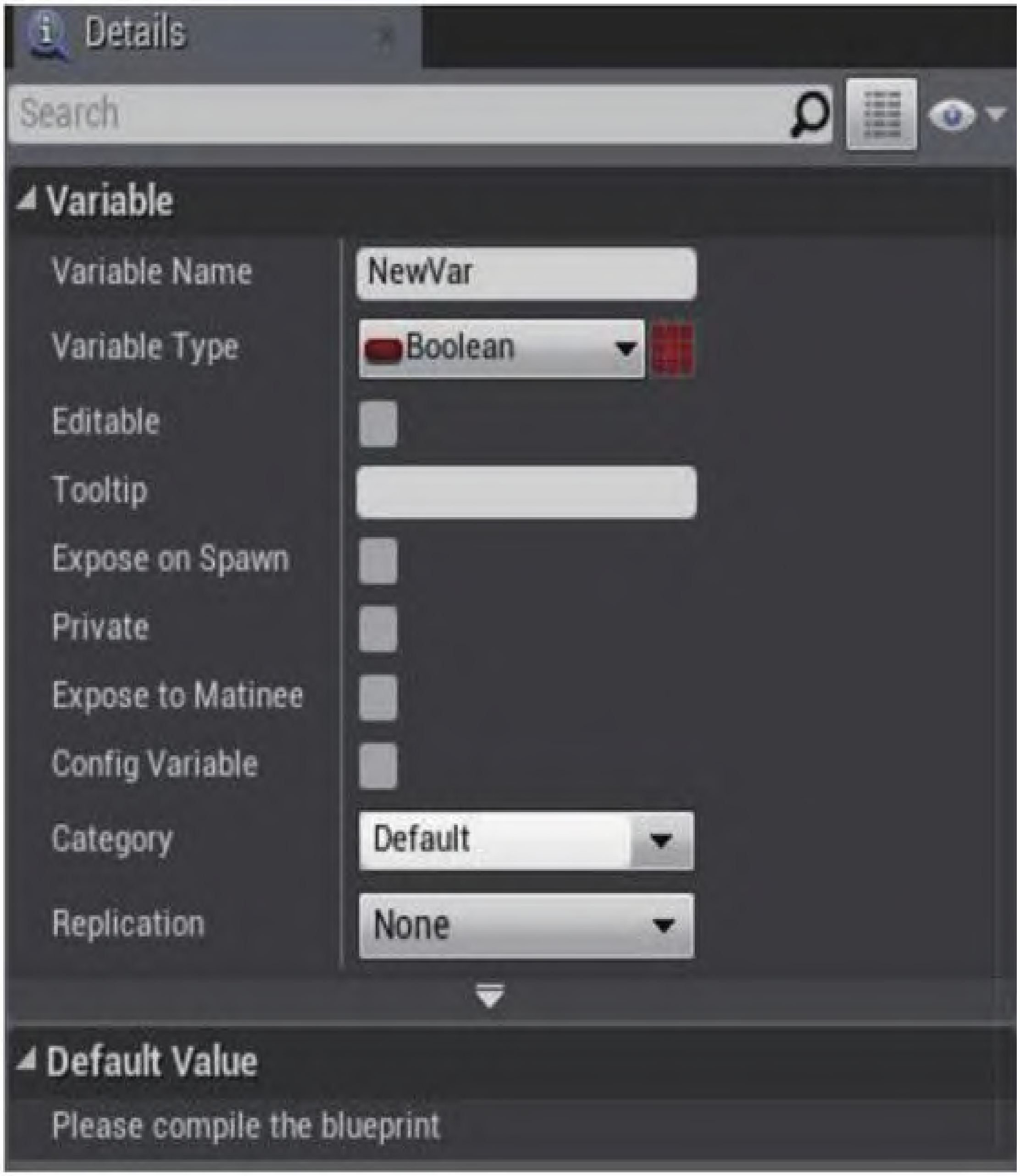



图3-187

## 1. 添加一个新变量

**Step 1** 在我的蓝图选卡内，在文本框中输入该变量的名称，并按下回车键，如图3-188所示。

**Step 2** 在详细信息选卡中，从Variable Type（变量类型）菜单中选择你想添加的变量的类型，如图3-189所示。

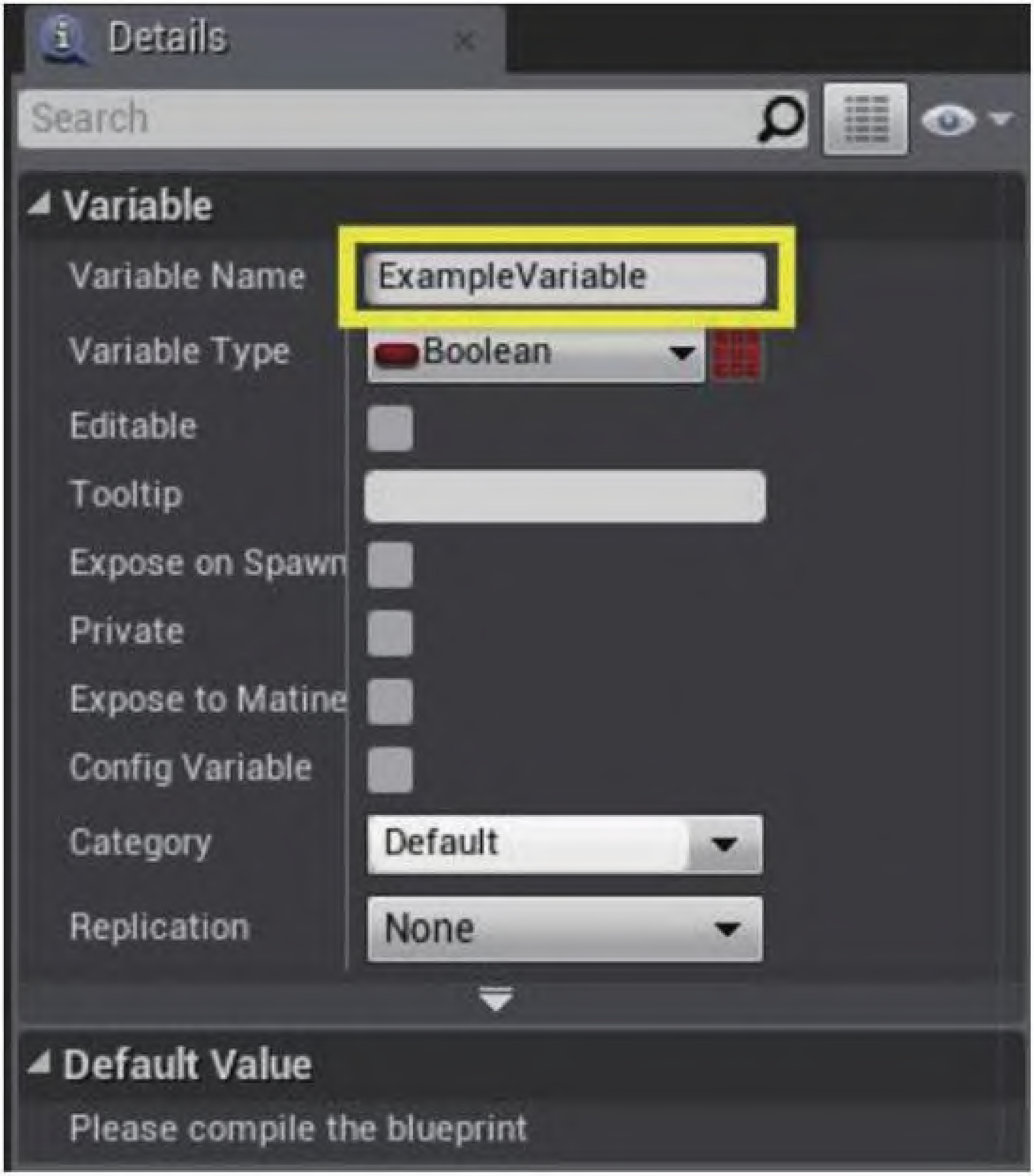

Details
Search
Variable
Variable Name
ExampleVariable
Variable Type
Boolean
Editable
Tooltip
Expose on Spawn
Private
Expose to Matine
Config Variable
Category
Default
Replication
None
Default Value
Please compile the blueprint

图3-188

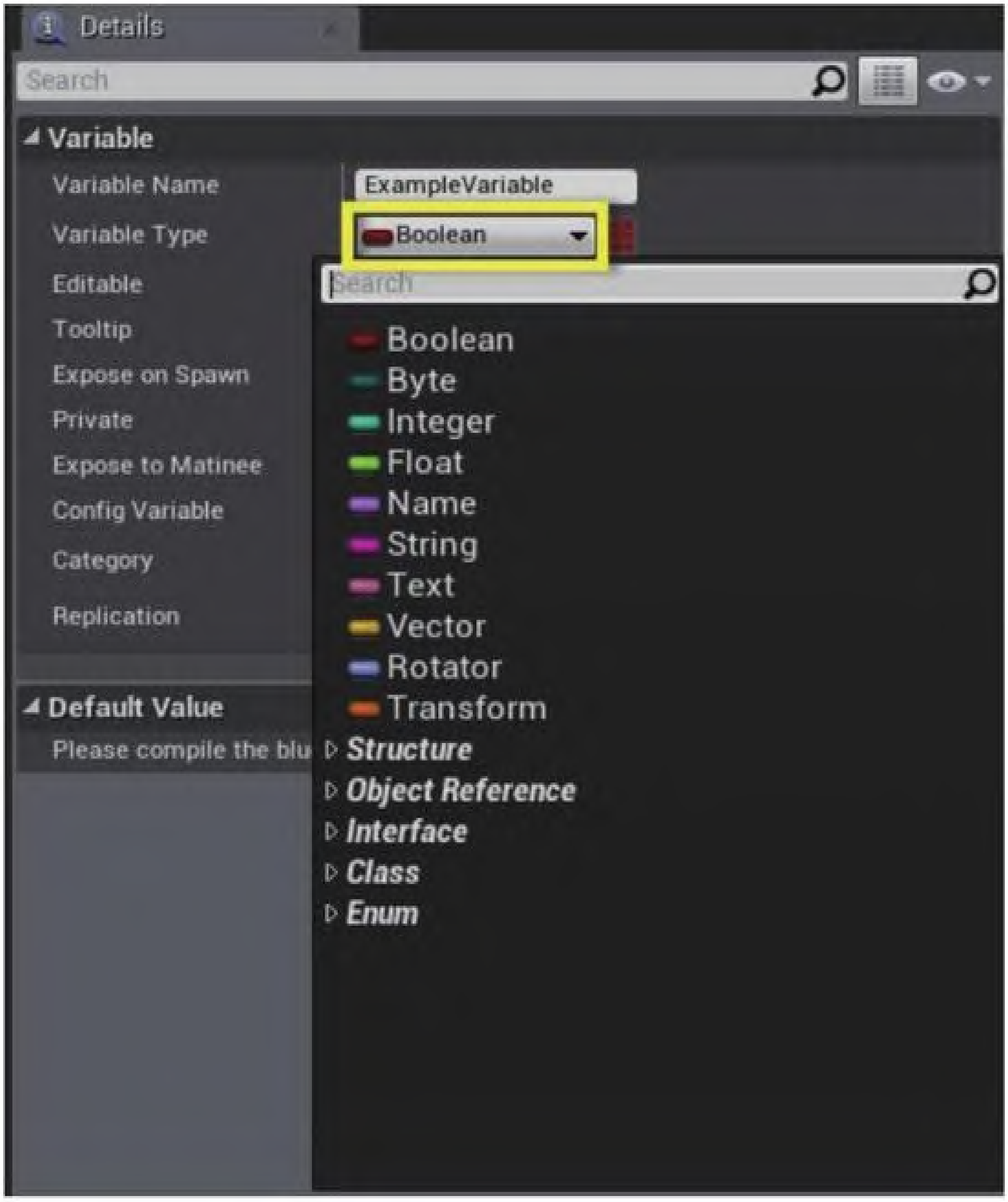



图3-189

某些变量类型需要展开来查看需要的子菜单项，比如对象或结构体，如图3-190所示。

**Step 3** 如果你正在创建这种变量类型的数组，请单击Variable Type（变量类型）下拉框旁边的Array（数组）按钮，如图3-191所示。

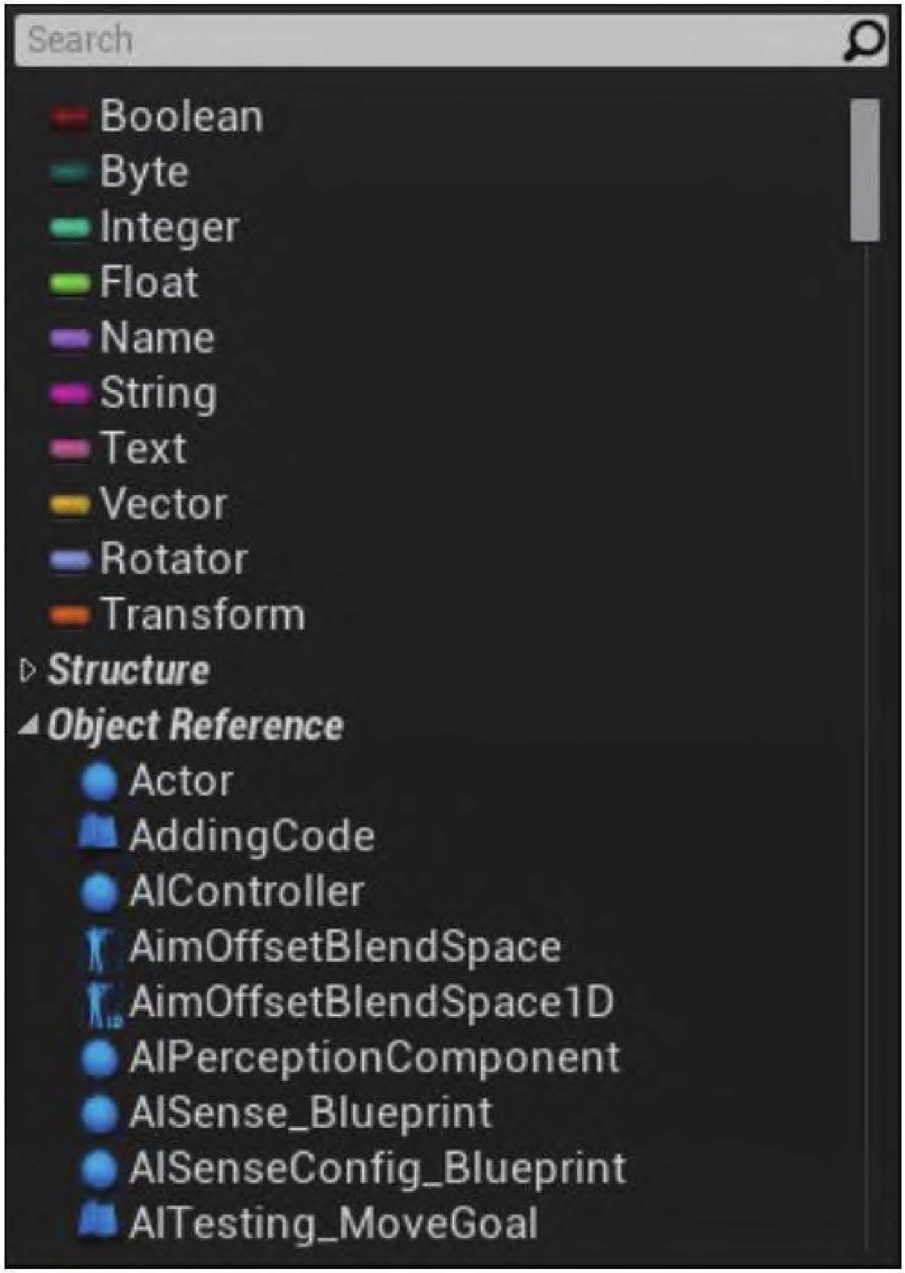

Search
Boolean
Byte
Integer
Float
Name
String
Text
Vector
Rotator
Transform
Structure
Object Reference
Actor
AddingCode
AIController
AimOffsetBlendSpace
AimOffsetBlendSpace1D
AIPerceptionComponent
AISense_Blueprint
AISenseConfig_Blueprint
AITesting_MoveGoal

图3-190

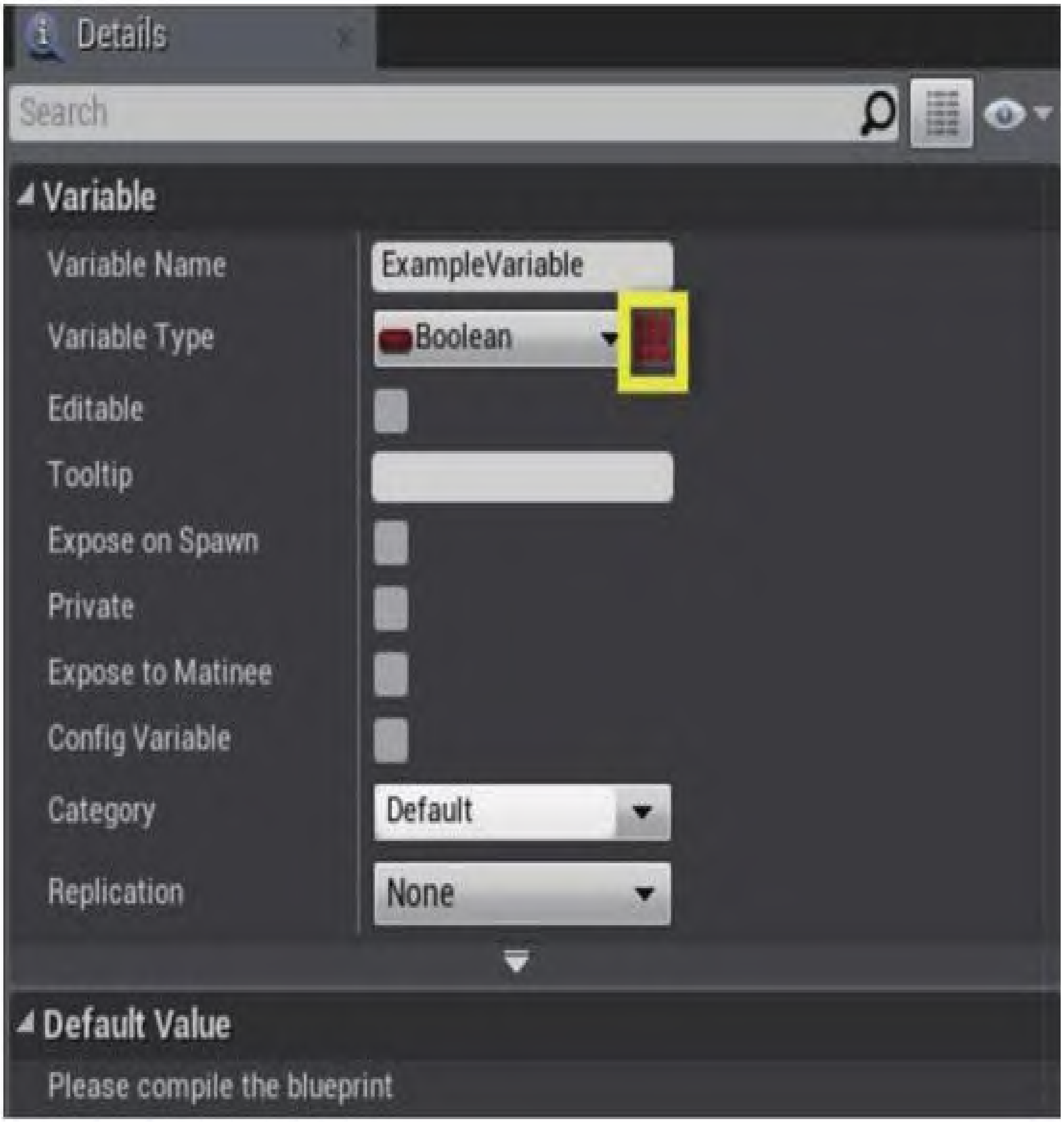



图3-191

### 2. 提升变量

你可以把一个节点上的任何输入或输出引脚提升为变量，除非该引脚是无类型的数组引脚，如图3-192所示。

**Step 1** 右击你想提升为变量的引脚，然后从弹出的菜单中选择Promote to Variable（提升为变量），如图3-193所示。

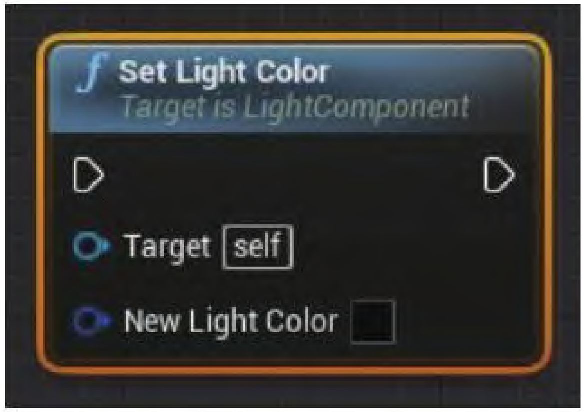



图3-192

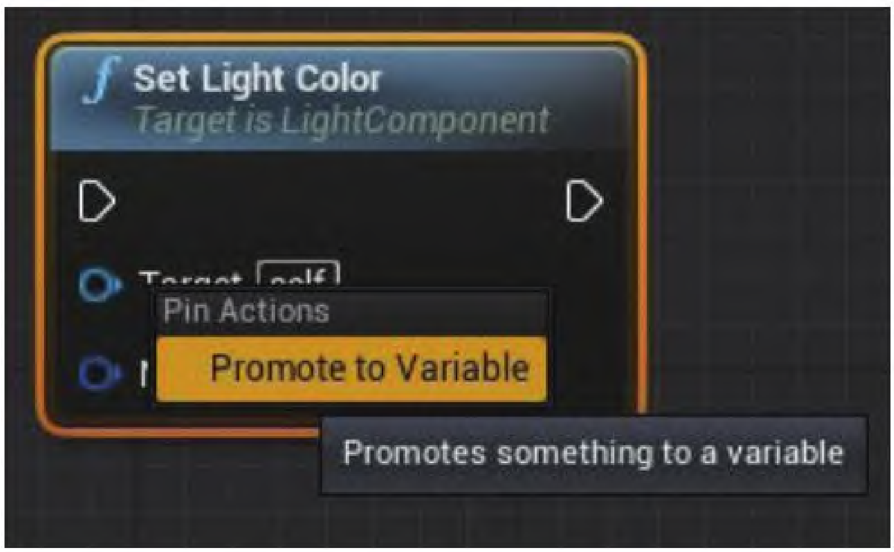



图3-193

**Step 2** 在新变量名称菜单中输入你的新变量的名称，如图3-194所示。

图3-194

图表上将出现那个新变量的引用。将一个输入引脚提升为变量将会建一个Get节点，将一个输出引脚提升为变量将创建一个Set节点，如图3-195所示。

图3-195

你的新变量将列出在我的蓝图选卡中，在那里你可以选择它并在Details（详细信息）选卡中编辑它的详细信息。

### 3.3.6.5 编辑变量

变量值可设置为蓝图节点网络的一部分，或者在执行前设置为默认值。

要想设置变量的默认值：

**Step 1**　在蓝图编辑器中选择Defaults（默认）模式或在Components（组件）或Graph（图表）模式中的Window（窗口）菜单内打开Blueprint Defaults（蓝图默认值）选卡，如图3-196所示。

**Step 2**　在你的变量名称的右侧输入想使用的默认值，如图3-197所示。

如果你在默认值列表中没有看到你的变量，那么请确定你单击了 按钮。

**1.重命名变量**

要想重命名一个变量：

**Step 1**　在我的蓝图选卡中右击变量名称，并在出现的菜单中选择Rename（重命名），如图3-198所示。

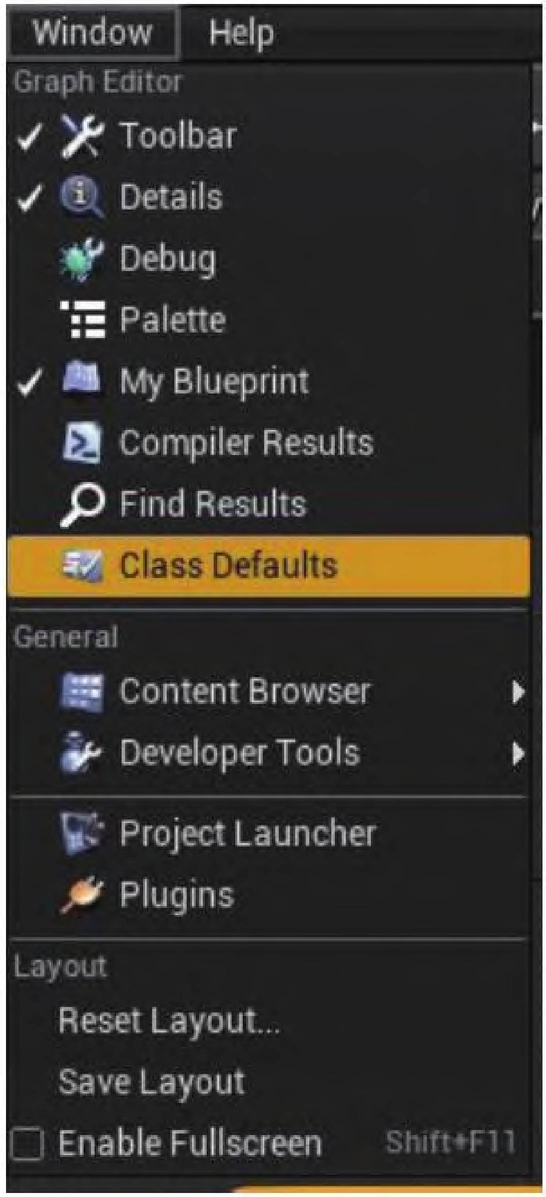

Window
Help
Graph Editor
Toolbar
Details
Debug
Palette
My Blueprint
Compiler Results
Find Results
Class Defaults
General
Content Browser
Developer Tools
Project Launcher
Plugins
Layout
Reset Layout...
Save Layout
Enable Fullscreen
Shift+F11

图3-196

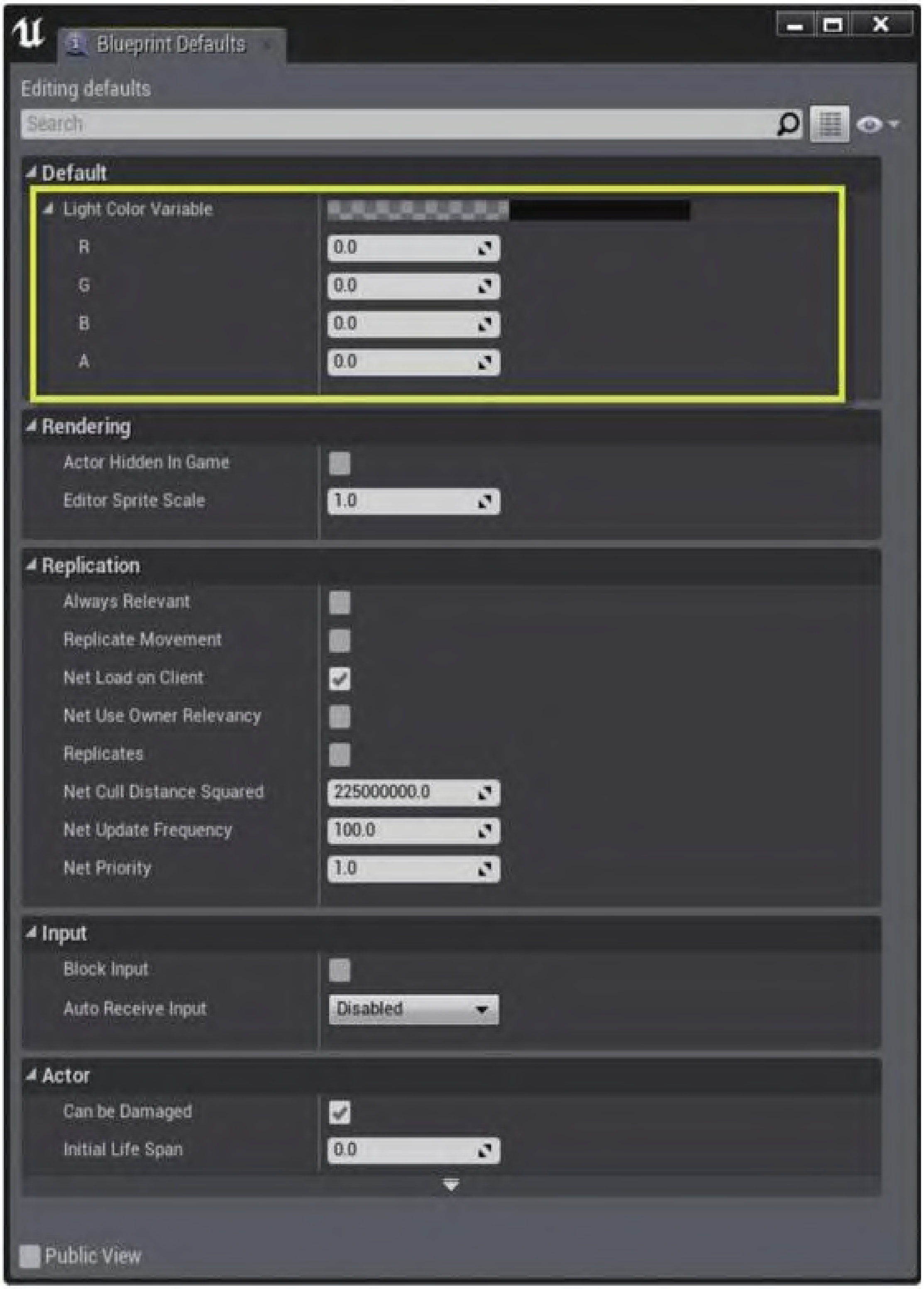

Blueprint Defaults
Editing defaults
Search
Default
Light Color Variable
R
0.0
G
0.0
B
0.0
A
0.0
Rendering
Actor Hidden In Game
Editor Sprite Scale
1.0
Replication
Always Relevant
Replicate Movement
Net Load on Client
Net Use Owner Relevancy
Replicates
Net Cull Distance Squared
225000000.0
Net Update Frequency
100.0
Net Priority
1.0
Input
Block Input
Auto Receive Input
Disabled
Actor
Can be Damaged
Initial Life Span
0.0
Public View

图3-197

**Step 2**　在文本框中输入新变量的名称，并按下回车键来保存它，如图3-199所示。

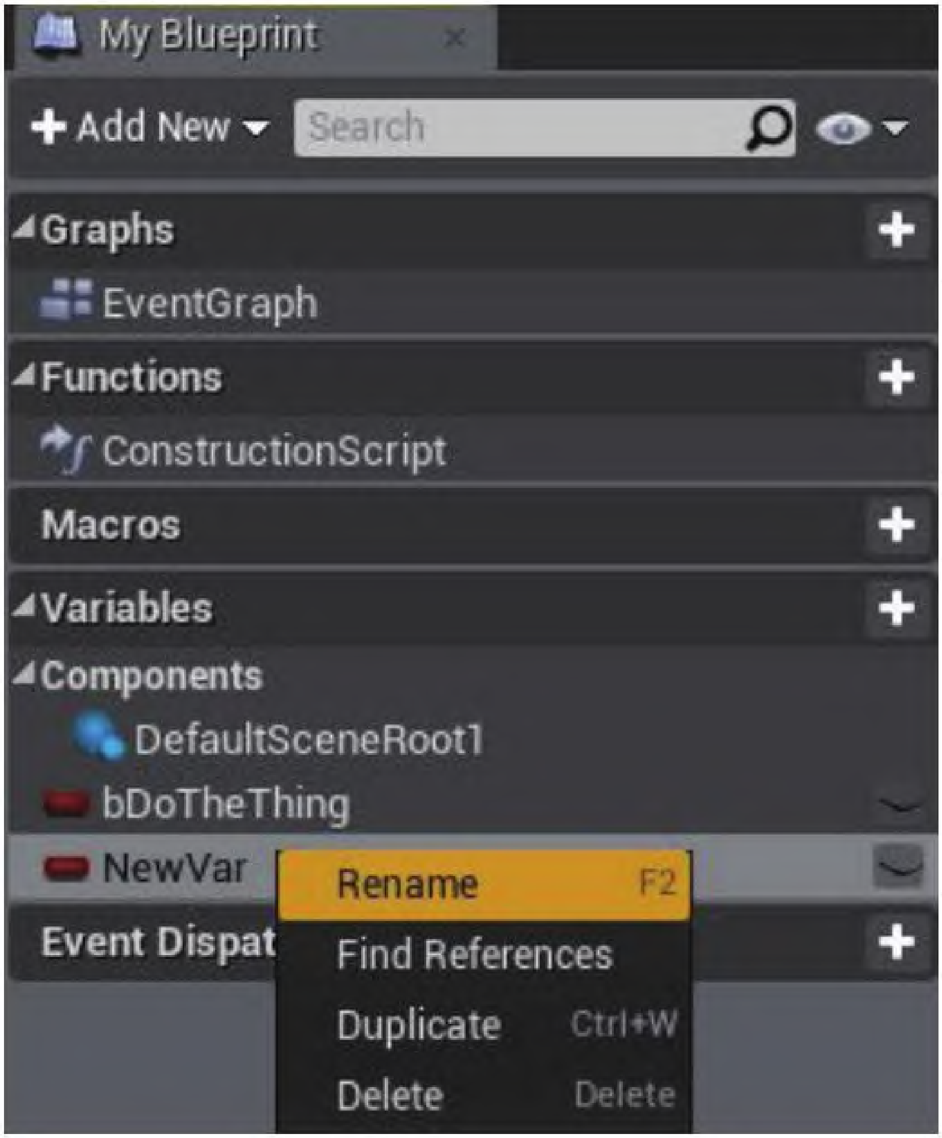



图3-198

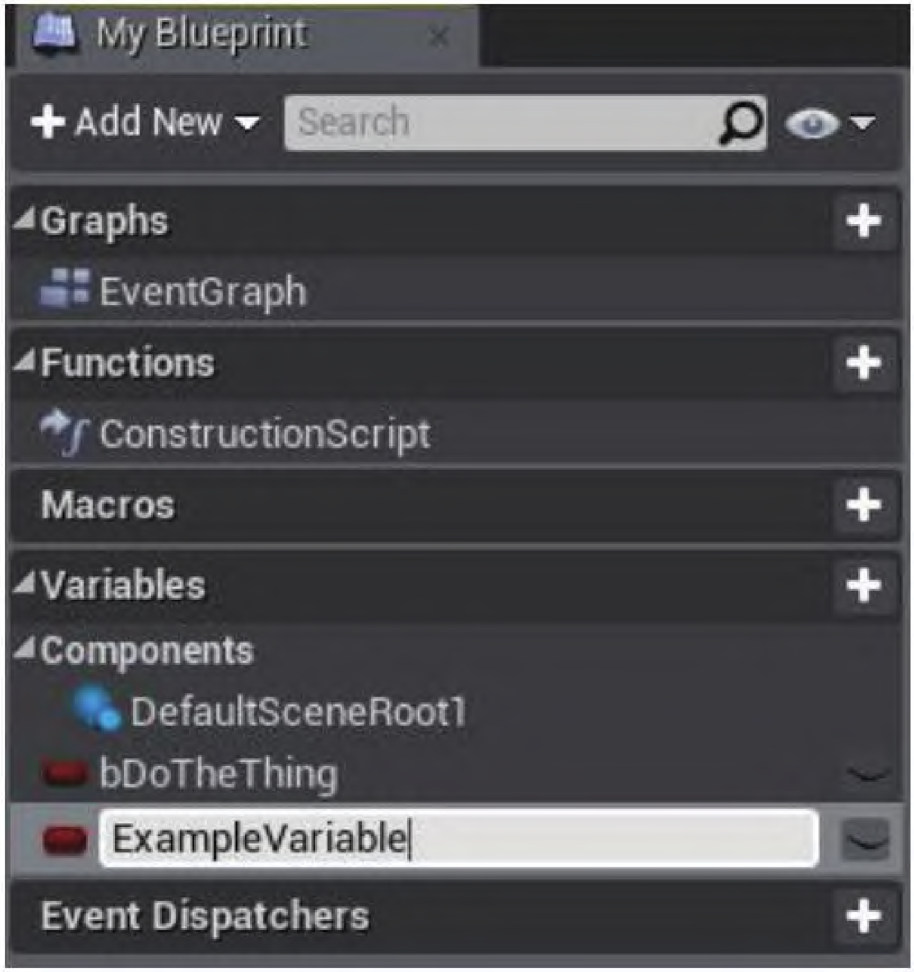



图3-199

## 2. 变量属性

你可以在详细信息面板中设置一个变量的所有属性。某些变量的属性数量会比这里显示的属性数量更多，比如向量的Expose to Matinee（暴露给Matinee）属性或像整型或浮点型这样的数值变量的Slider Range（滑块范围）属性，有关变量的属性可参见下表。

| 属　性 | 描　述 |
|---|---|
| Variable Type（变量类型） | 设置下拉菜单中的变量类型，并决定该变量是否是个数组 |
| Editable（可编辑） | 设置是否可以在 Blueprint Defaults（蓝图默认值）及蓝图的详细信息面板中编辑该变量的值 |
| Tooltip（工具提示信息） | 设置该变量的工具提示信息 |
| Private（私有） | 设置该变量是否应该是私有变量，如果是私有的，则不能被该蓝图的子类所修改 |
| Category（类目） | 选择一个现有类目或者输入一个新类目的名称。设置类目决定了在蓝图默认值、我的蓝图选卡及蓝图的详细信息选卡中该变量出现的位置 |
| Replication（复制） | 选择是否应该在客户端之间复制该变量的值，是否复制该值，是否应该通过调用函数产生一个通知 |

### 3.获取和设置变量值

变量也可以作为你的蓝图网络的一部分来进行编辑，通过Get和Set节点来完成。创建这些节点的最简单的方法是，从变量选卡中直接拖动变量到事件图表中，将出现一个小菜单，询问你是否要创建一个Get或Set节点，如图3-200所示。

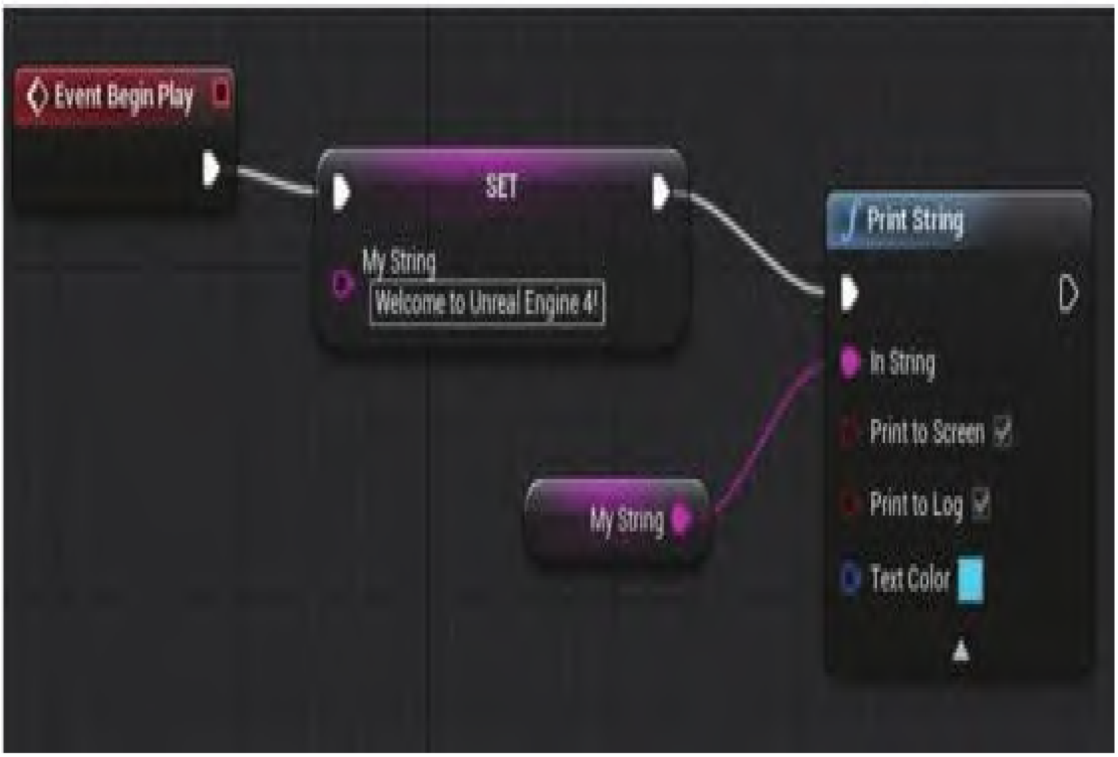



图3-200

Get节点：

Get节点向部分节点网络提供一个变量值。一旦创建这些节点，可以把它们连入适当类型的任何节点中，如图3-201所示。

Set节点：

Set节点允许你修改变量的值。注意，这些节点必须通过执行引脚连线进行调用才能执行，如图3-202所示。

图3-201

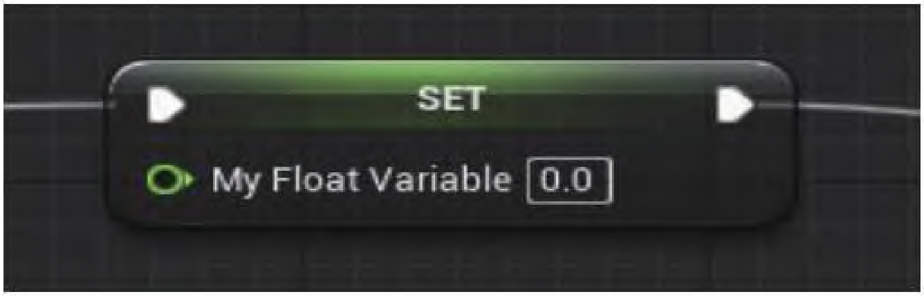



图3-202

| 当从我的蓝图选卡中进行拖动时可以使用的快捷方式 | |
|---|---|
| Ctrl—拖动 | 创建一个Get节点 |
| Alt—拖动 | 创建一个Set节点 |

## 3.3.7 直接蓝图通信

本节面讲述使用直接蓝图通信在两个蓝图之间进行通信和信息传递。

在此例中，玩家将通过箱子传递信息，启用一个粒子特效（并与其形成通信）。

此指南使用启用新手内容的Blueprint Third Person模板。

**Step 1** 在Content Browser中打开Content/StarterContent/Shapes文件夹。

**Step 2** 右击Shape_Cube按钮，然后在Asset Actions下选择Create Blueprint Using This如图3-203所示。

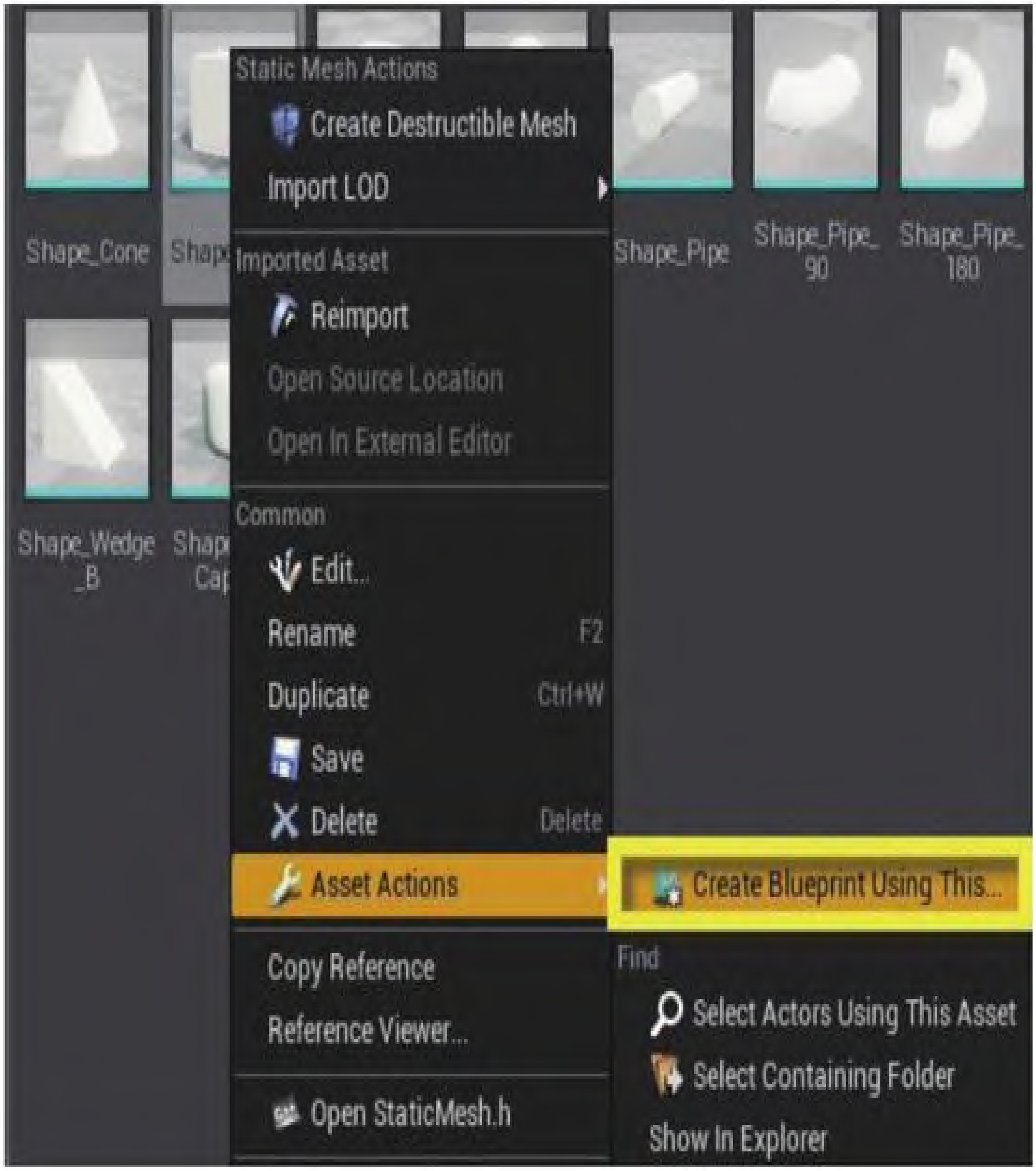



图3-203

在创建蓝图的确认框上单击Ok按钮。

**Step 3** 在Shape_Cube蓝图中，选择Components窗口左上角的Static Mesh。

**Step 4** 在Details面板中，将Collision Presets改为OverlapOnlyPawn，如图3-204所示。

此设置可确保只有Pawns和扩展而来的角色才可通过方块。

**Step 5**　在MyBlueprint窗口中，单击Add Variable按钮，将变量命名为TargetBlueprint，如图3-205所示。

这便是需要直接进行通信的另一个蓝图。

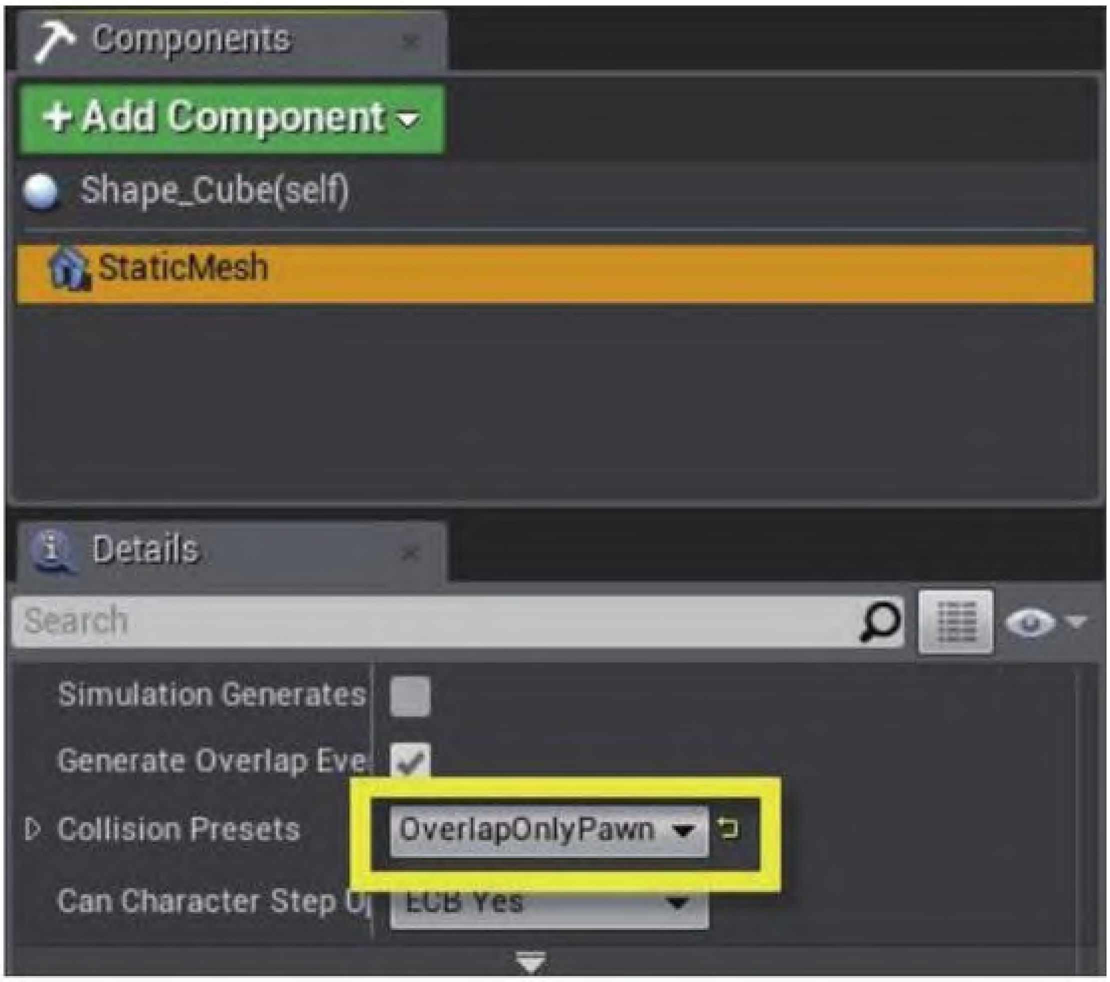



图3-204

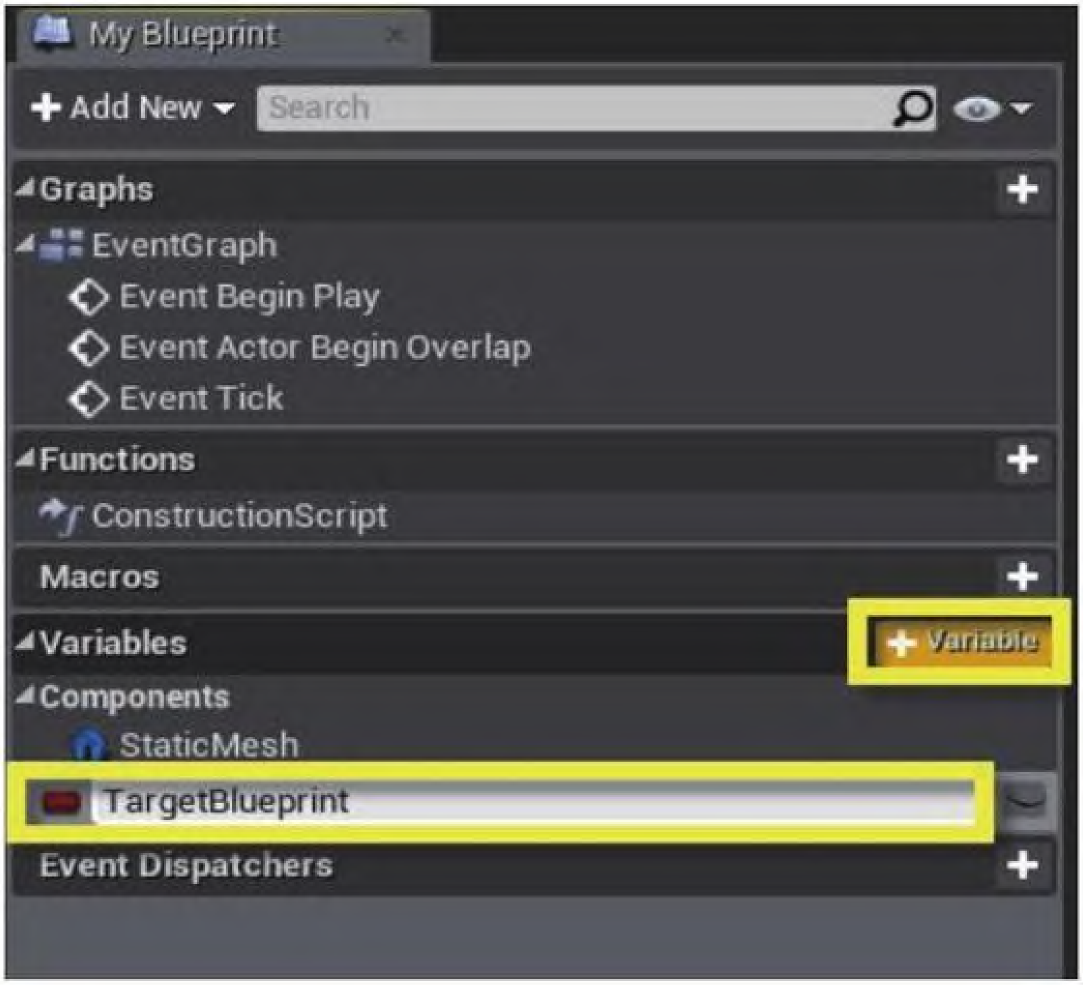



图3-205

**Step 6** 在此变量的Details面板中，将Variable Type设为Blueprint_Effect_Fire，并勾选Editable，如图3-206所示。

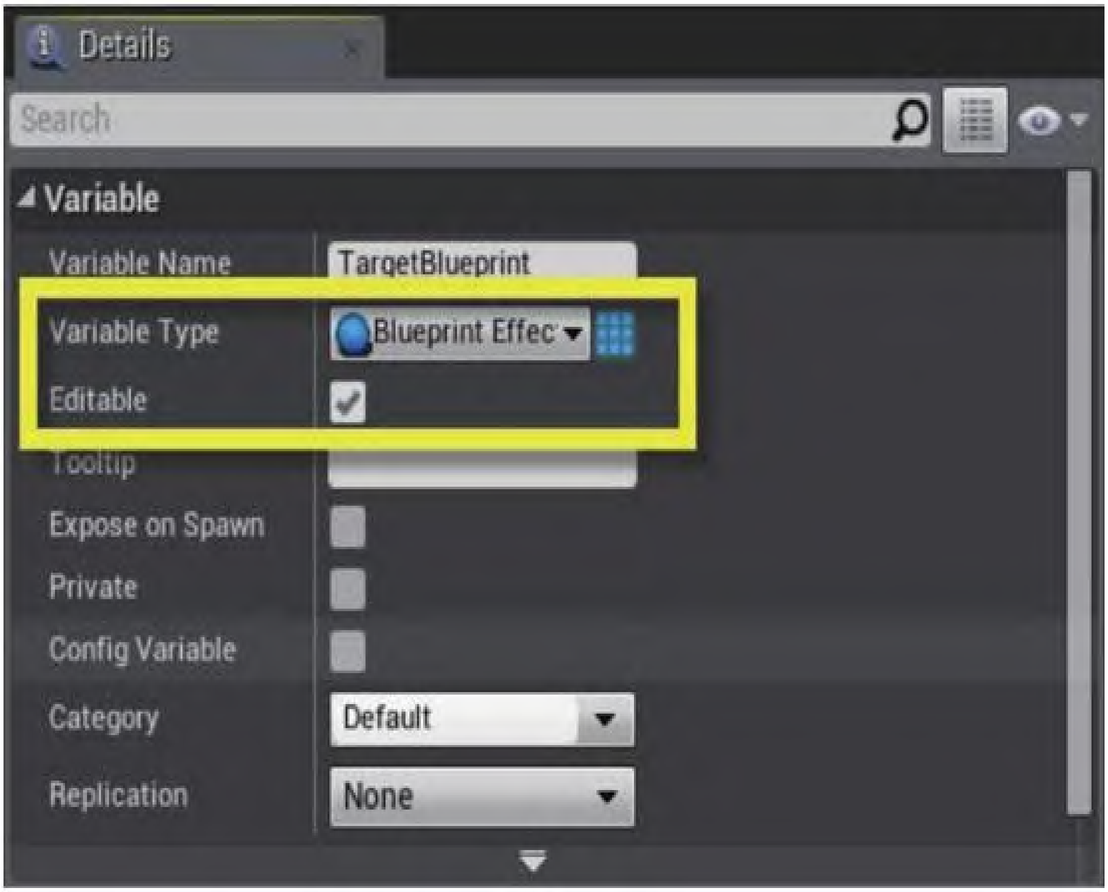



图3-206

我们可以在此对需要进行通信的蓝图进行设置，并将变量设为可公开编辑（在蓝图外进行编辑）。

**Step 7** 在My Blueprint窗口中选择Static Mesh，然后在Details面板中添加一个On Component Begin Overlap事件，如图3-207所示。

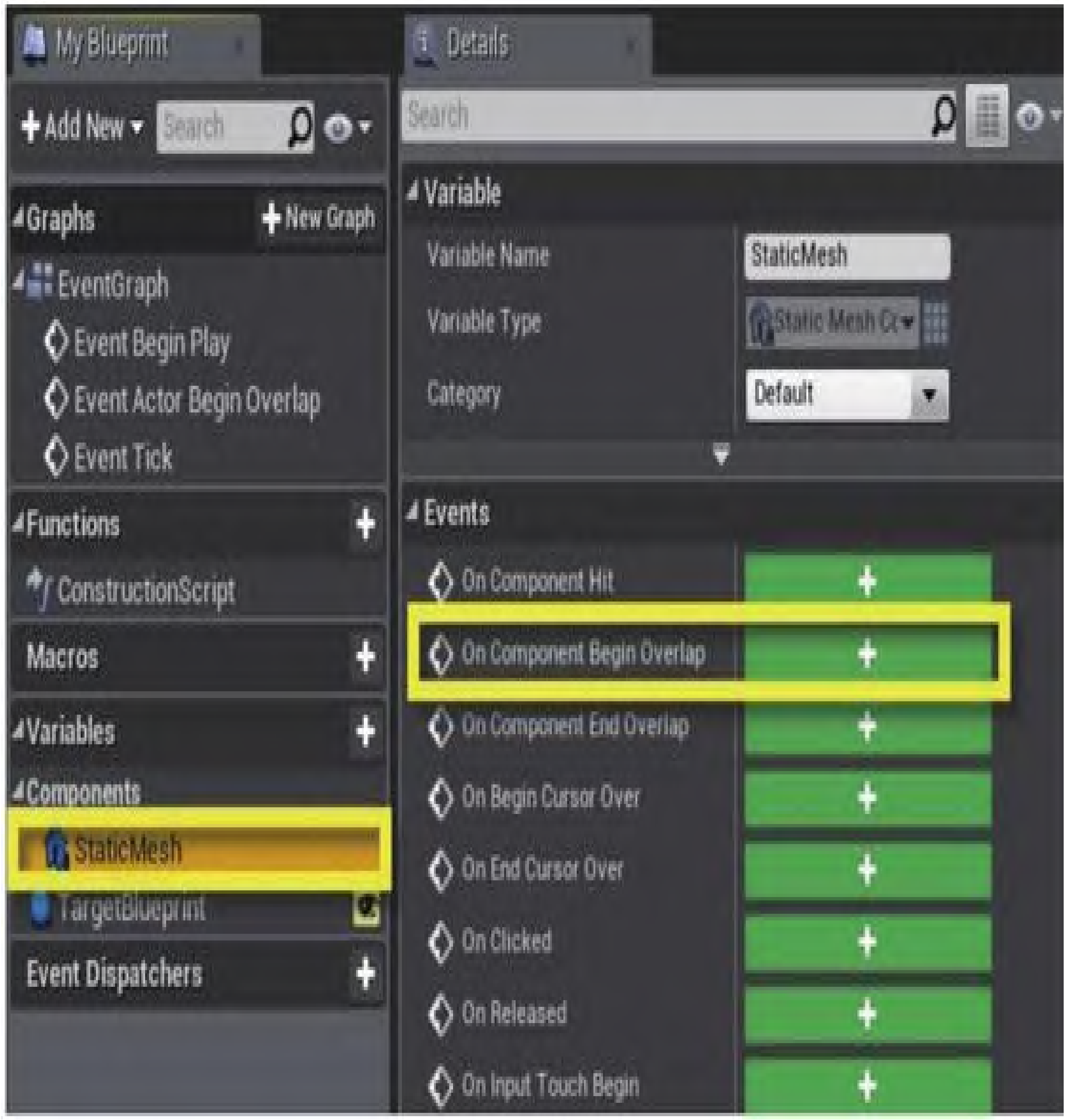



图3-207

此操作将添加一个节点并打开事件图表，在此可提供进入方块时发生的脚本。

**Step 8**　将Target Blueprint的引出连线连接到Get P Fire和Get Fire Audio，然后再连接到Activate节点，如图3-208所示。

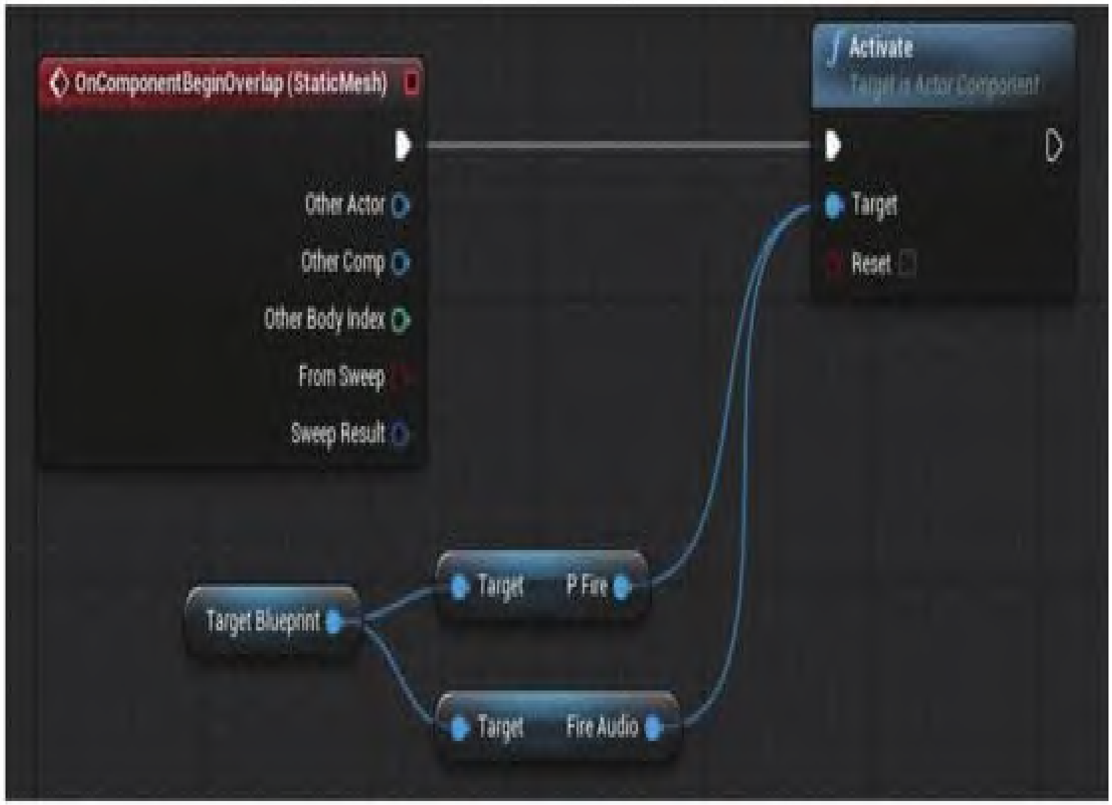



图3-208

我们可以在目标蓝图中访问粒子效果和音频组件，并将它们启用。

**Step 9** 在Content Browser的Blueprints中，将Shape_Cube蓝图拖入关卡，如图3-209所示。

图3-209

**Step 10**　打开内容浏览器中Content/StarterContent/Blueprints文件夹下的Blueprint_Effect_Fire蓝图。

**Step 11**　在Components窗口中选择P_Fire，然后在Details面板中取消勾选Auto Activate，如图3-210所示。

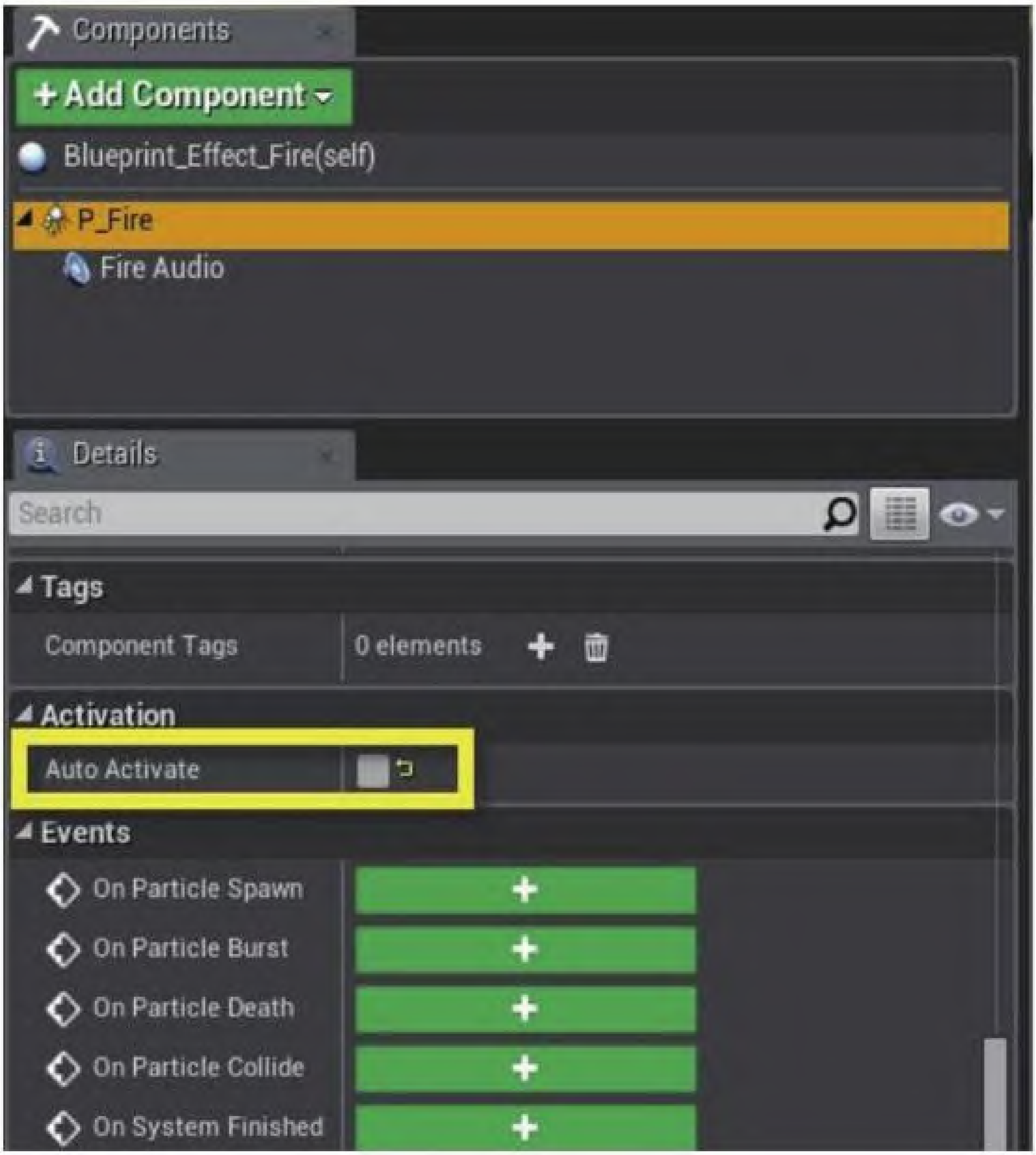



图3-210

我们不需要此特效自动开启。进入关卡中的方块后才会对其发出开启指令。

**Step 12**　将Blueprint_Effect_Fire蓝图拖入关卡。

**Step 13**　在Details面板中选择Shape_Cube蓝图，使用下拉菜单选择Blueprint_Effect_Fire蓝图，如图3-211所示。

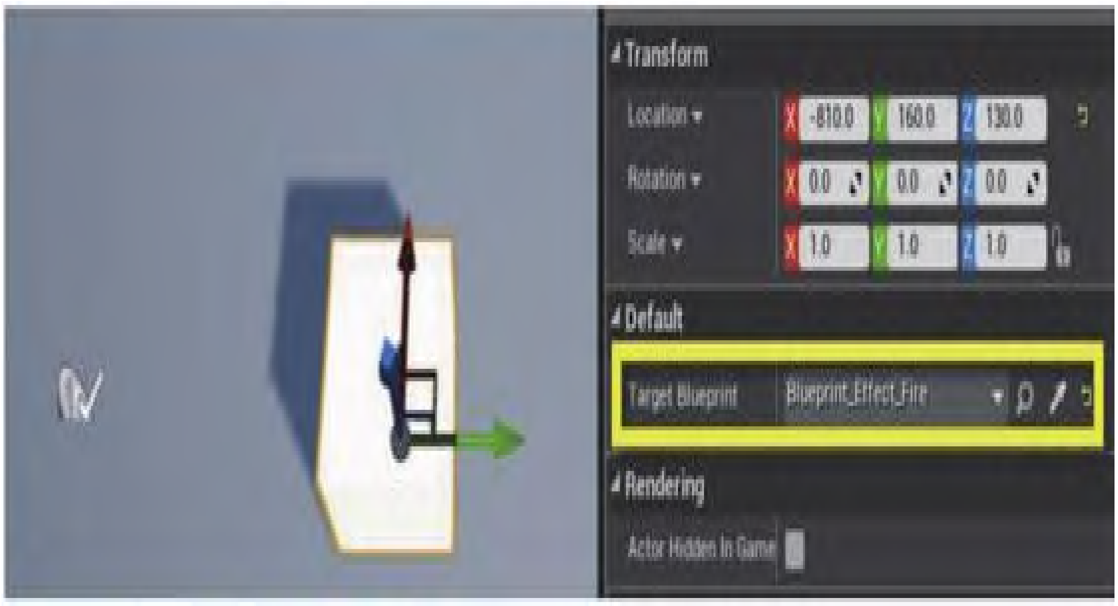



图3-211

此操作将在关卡中指定需要进行通信的Blueprint_Effect_Fire蓝图实例。如关卡中放置有数个Blueprint_Effect_Fire蓝图实例，每个实例都将显示在下拉菜单中，以便确定哪个实例是Target Bluperint进行通信的对象。

单击滴管图标即可选择关卡中的Target Bluperint实例，无须使用下拉菜单。注意：只能基于指定的变量类型选择目标蓝图。

**Step 14**　单击Play按钮即可在编辑器中测试角色穿过方块。

角色进入方块后，关卡中将出现火焰特效。使用直接蓝图通信不仅可以从一个蓝图中变更另一个蓝图的属性，还可传递变量值、调用函数或事件，或执行蓝图通常可进行的其他操作。

## 3.3.8　蓝图转换范例

此节讲述如何使用蓝图转换与不同的蓝图进行交流并共享信息。在此例中玩家可对物体进行射击，每次命中后均会使其变小。每次命中物体时还将

告知角色蓝图并增加玩家的移动速度。

此指南使用启用新手内容的Blueprint First Person模板。

**Step 1** 在视口中选择一个EditorCube网格体，然后在Details面板中单击Blueprint/Add Script按钮，如图3-212所示。

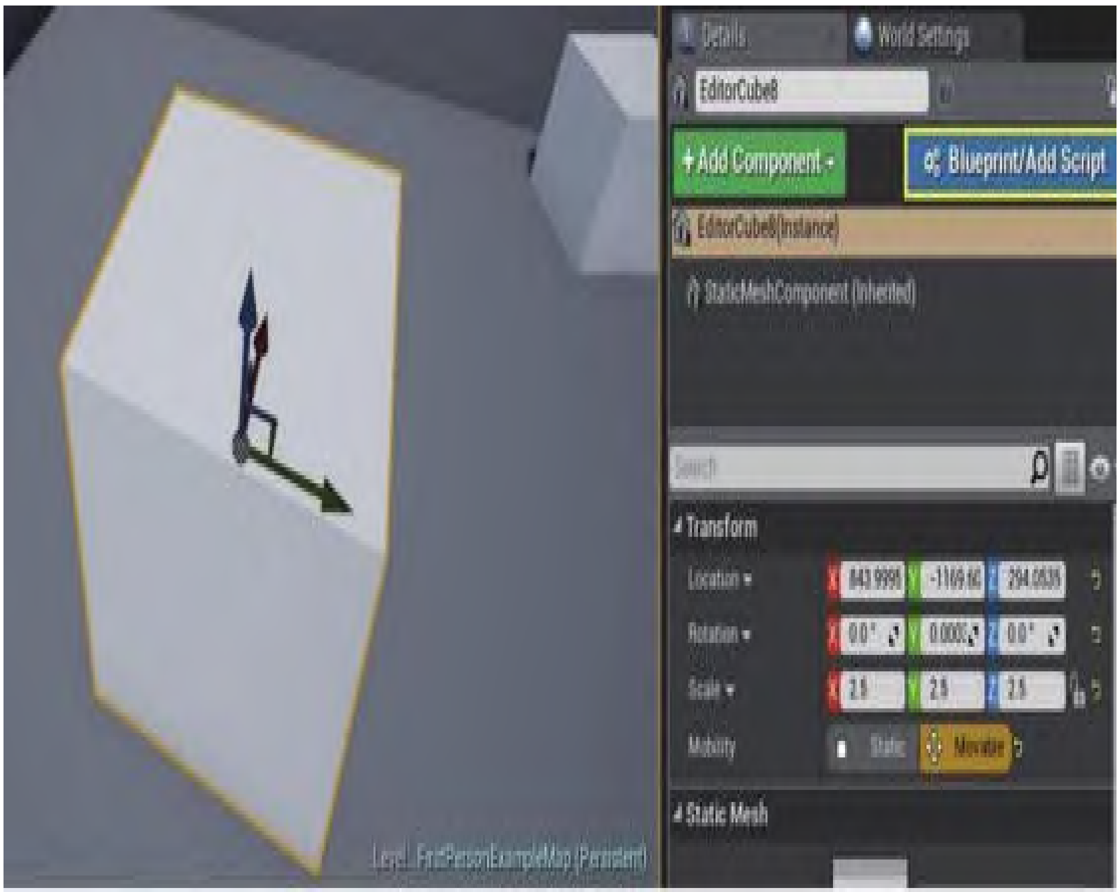



图3-212

**Step 2** 确认窗口出现后，将命名改为LargeCube_Blueprint，单击Ok按钮重命名并创建蓝图。

**Step 3** 打开内容浏览器中Content/First PersonBP/Blueprints文件夹下的FirstPersonProjectile蓝图，如图3-213所示。

图3-213

这是游戏进程中玩家武器开火时转换物的蓝图。

**Step 4**　找到Event Hit节点，拖动Other引脚的引出连线，然后搜索并添加Cast To LargeCube_Blueprint节点，如图3-214所示。

图3-214

在Event Hit节点上，Other引脚代表命中的其他Actor。使用Cast To节点可进行检查，确定LargeCube_Blueprint是否即为命中的其他Actor。如是，即可对其进行访问，并在蓝图中执行事件或函数、变更其中的变量值，或变更蓝图中的组件属性。

**Step 5** 在As Large Cube Blueprint引脚引出连线后，使用Get Static Mesh Component和Set World Scale 3D，然后以如图3-215所示的方式进行连接。

图3-215

在Add Impulse at Location之后，将转换到Large Cube蓝图并获取其中的静态网格体（实际的立方体），然后再为其设置一个新的World Scale 3D值。当前New Scale设为0，然而每次命中Large Cube蓝图后，其尺寸均会缩小。

**Step 6** 在Static Mesh Component引出连线后，使用Get World Scale并连接到一个Vector * Float引脚（设为0.5），然后插入New Scale，如图3-216所示。

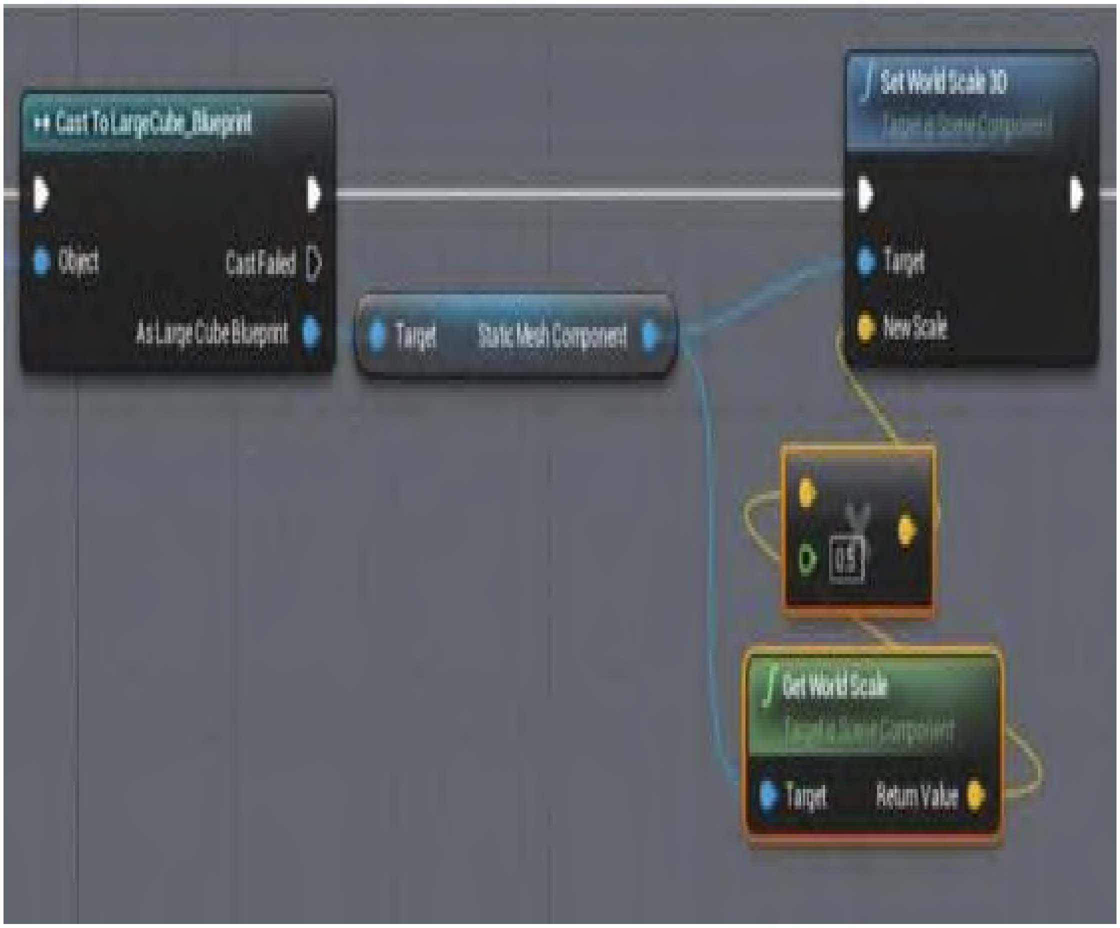



图3-216

此操作将获取立方体当前的世界场景比例，并在每次命中时将其乘以0.5（或将其当前大小减半），以缩小其大小。

**Step 7**　右击并添加一个Get Player Character节点，之后从Return Value连出引线到Cast To FirstPersonCharacter，如图3-217所示。

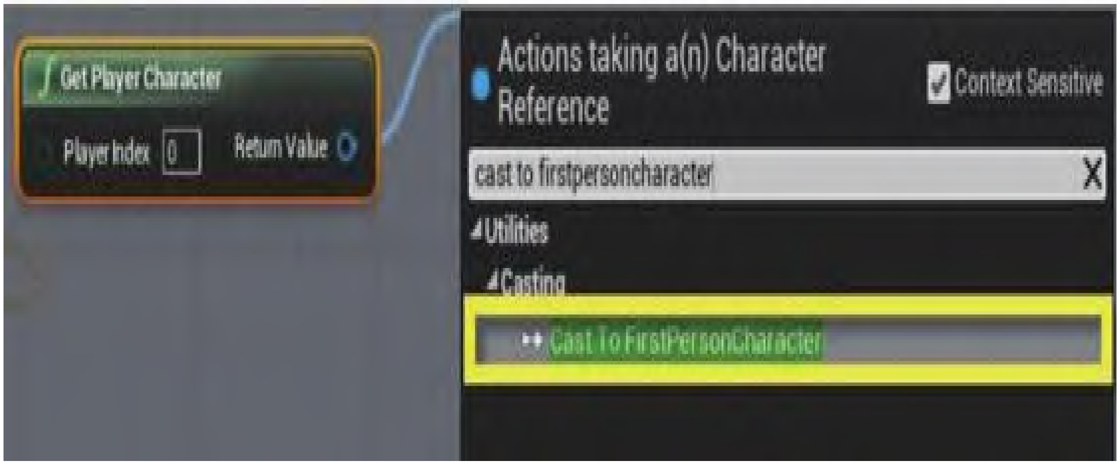



图3-217

此处检查玩家角色是否使用FirstPersonCharacter蓝图（此模板默认的蓝图）。执行此操作后即可访问玩家角色的事件、函数、变量或属性并进行修改。

**Step 8**　将Set World 3D节点和Cast To FirstPersonCharacter节点连接起来。

**Step 9**　在As First Person Character引脚之后添加Get Character Movement节点，如图3-218所示。

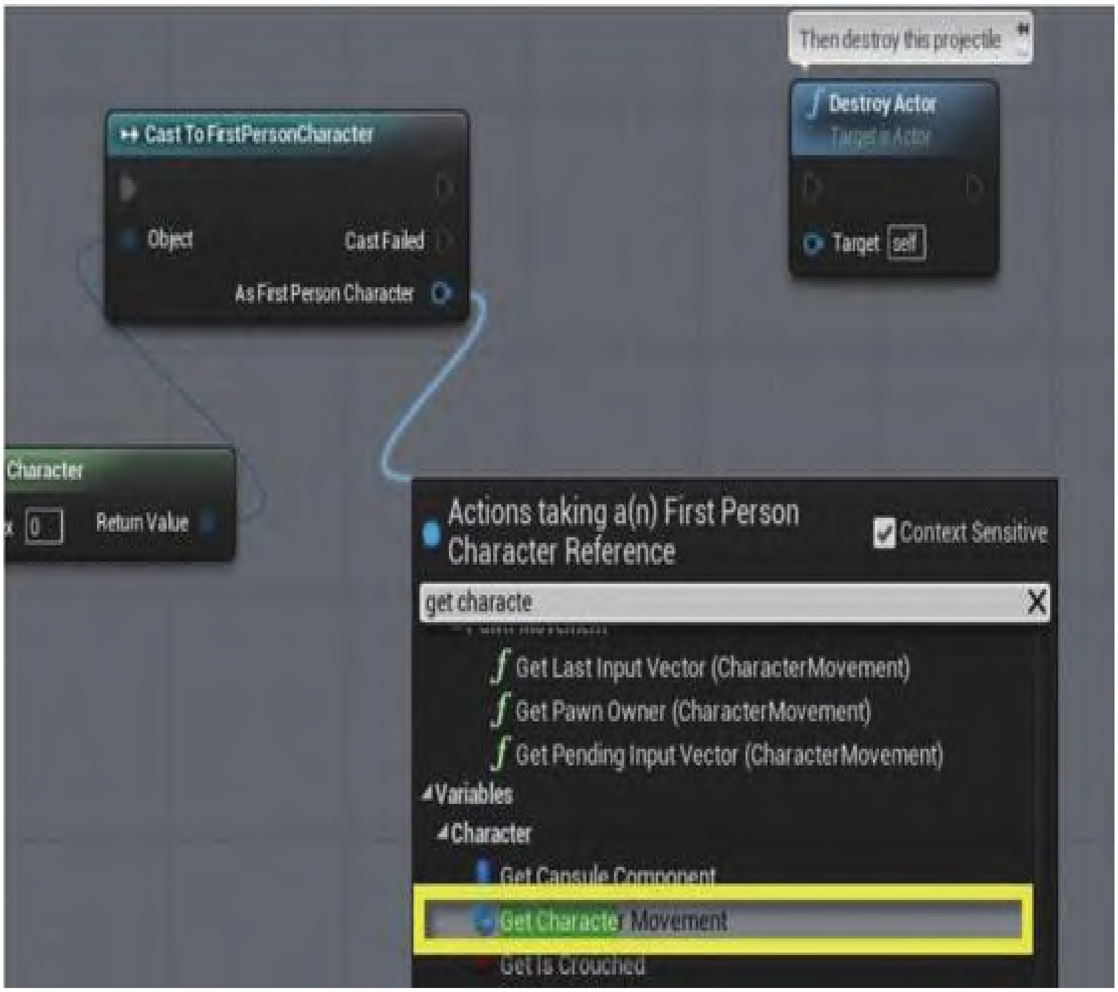



图3-218

命中Large Cube后将对角色的速度进行调整，使角色加快奔跑速度。

**Step 10** 在Character Movement节点引出连线后，添加Set Max Walk Speed和Get Max Walk Speed节点。

**Step 11** 在Get Max Walk Speed节点引出连线后，使用Float+Float节点（设为200），然后将其和Set Max Walk Speed引脚连接起来，如图3-219所示。

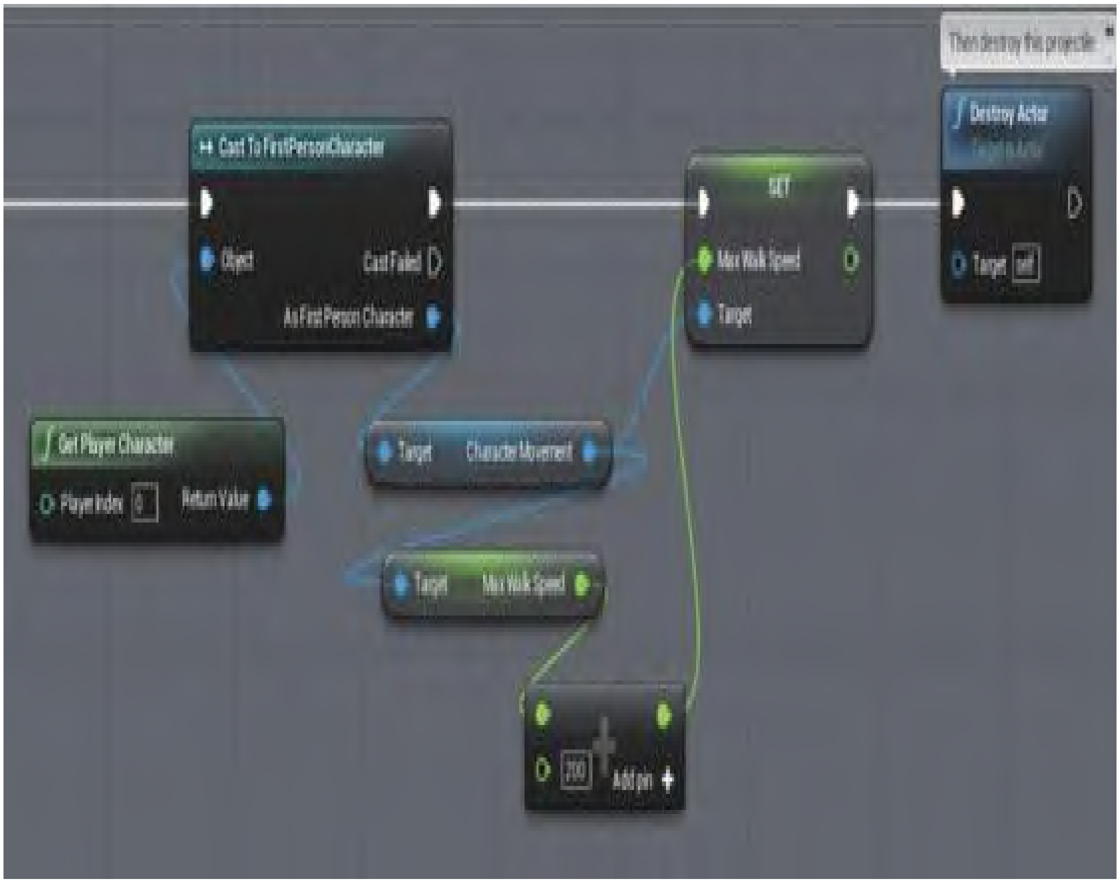



图3-219

将节点连接后，每次命中立方体时玩家角色的移动速度便会加快。

**Step 12**　将两个Cast To节点的Cast Failed引脚和Destroy Actor节点连接起来，如图3-220所示。

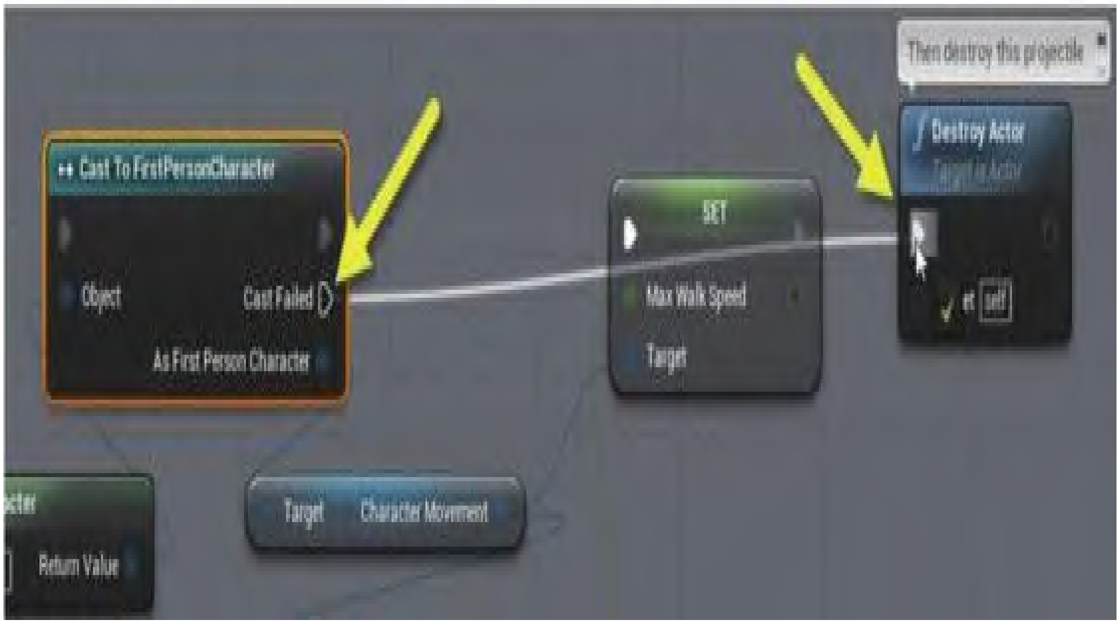



图3-220

当玩家命中的对象不是Large Cube时，仍需调用Destroy Actor节点销毁转换物。当命中Large Cube的不是First Person Character时，仍需销毁转换物。

Step 13　转换物蓝图的结束顺序和图3-221所示相似。

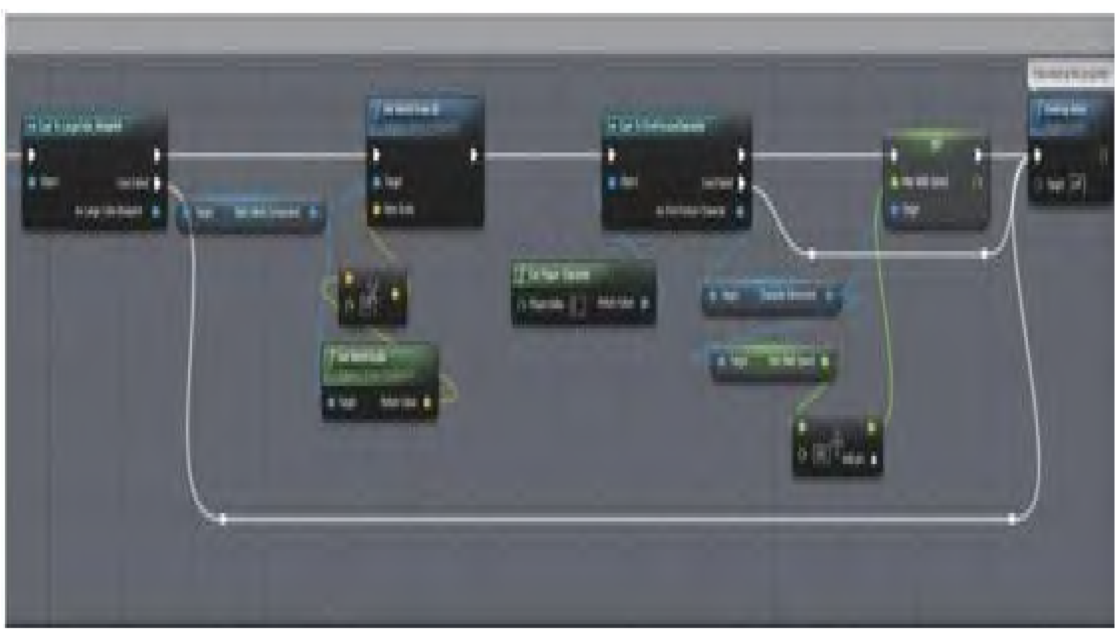


图3-221

为便于查看，我们使用Reroute Nodes对线条的分布进行了调整，但连接方式须保持一致。

**Step 14**　单击Compile按钮和Save按钮，然后单击Play按钮在编辑器中进行测试。

每次命中Large Cube蓝图时，其均会变小，角色移动速度将加快。

## 3.3.9　事件分配器

事件分配器被调用时，也将执行实施事件且事件与事件分配器绑定的全部蓝图。换言之，事件分配器被调用时，与其绑定的所有事件也将被调用，以便从单一源同时执行多个事件。

此节将说明如何设置事件分配器并将事件绑定到事件分配器。

在此例中，玩家按键后灌木丛中将出现火焰粒子效果。

此指南使用启用新手内容的Blueprint Third Person模板。

**Step 1**　在内容浏览器中的Content/ThirdPersonBP/Blueprints文件夹下，打开Third PersonCharacter蓝图，如图3-222所示。

**Step 2**　单击My Blueprint窗口中的Add Event Dispatcher按钮并将其命名为StartFire，如图3-223所示。

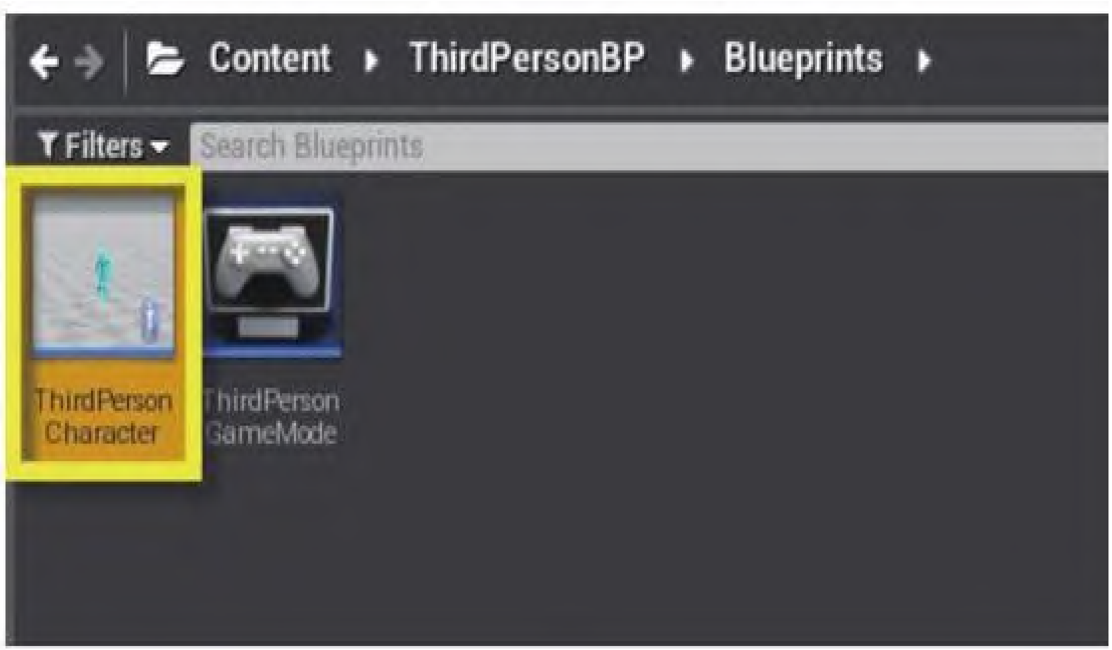


图3-222

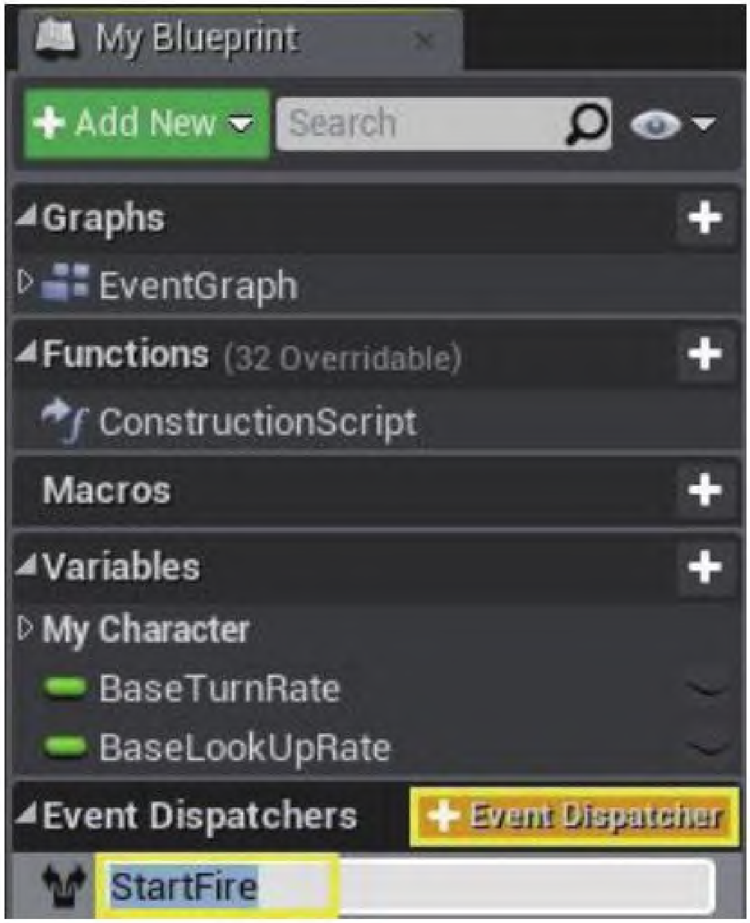



图3-223

**Step 3**　在图表窗口中右击，添加一个F按键事件。

**Step 4**　将StartFire事件分配器拖入图表并选择Call，如图3-224所示。

**Step 5**　将F按键事件和Call StartFire事件分配器连接起来，如图3-225所示。

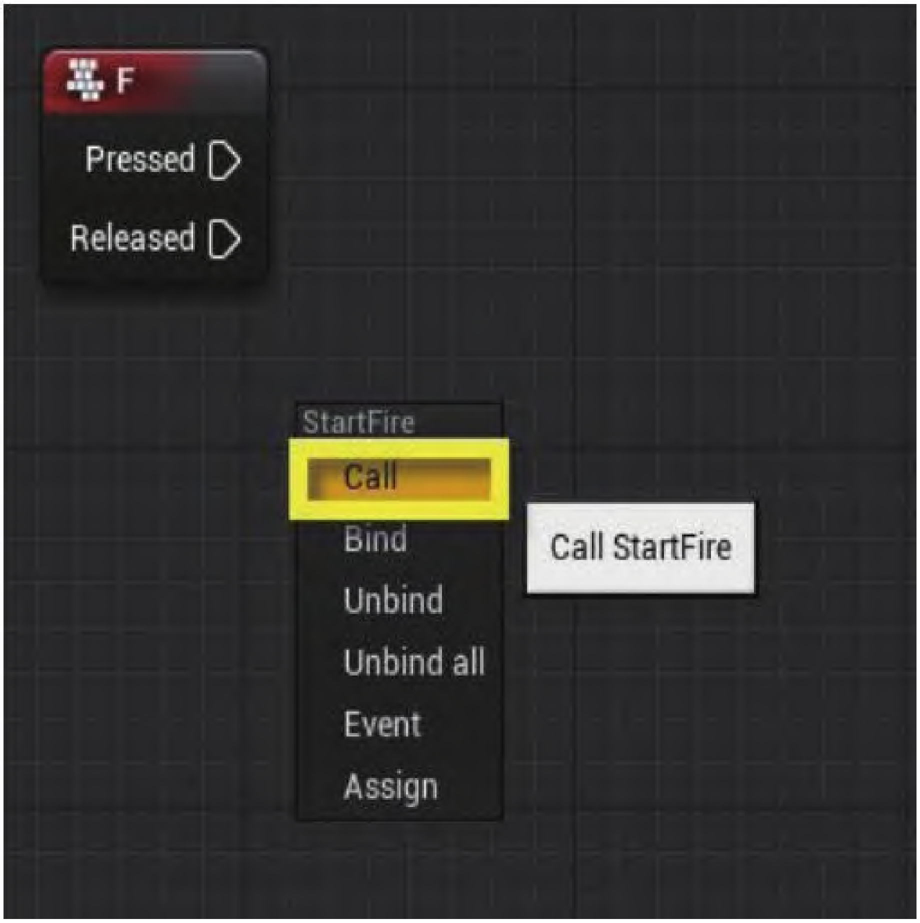



图3-224

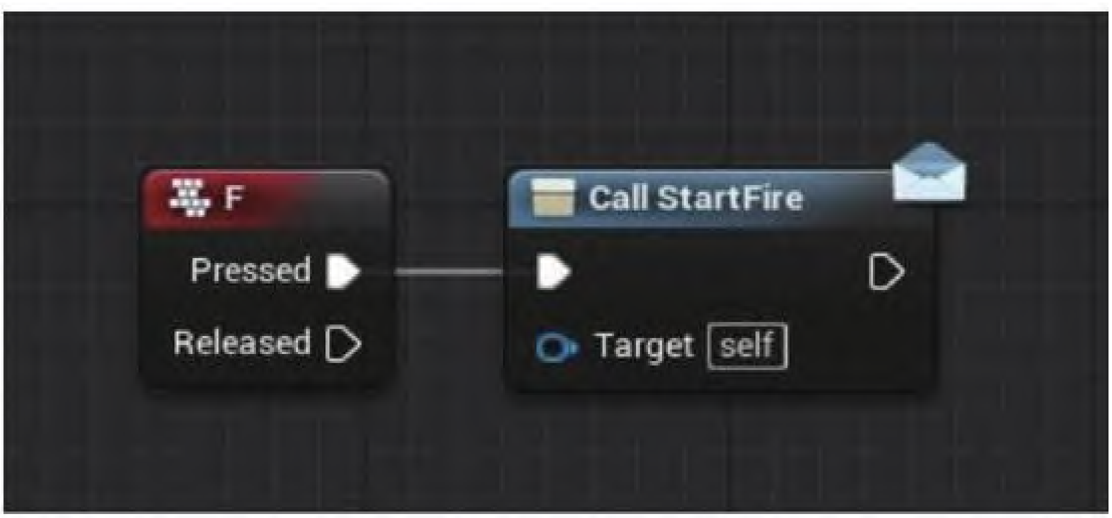



图3-225

F键按下时，事件与此事件分配器绑定的蓝图也将被执行。

**Step 6**　编译并保存，然后关闭蓝图。

**Step 7**　在Content Browser的Content/StarterContent/Props中，将SM_Bush蓝图拖入关卡，如图3-226所示。

图3-226

**Step 8**　单击主工具栏的Blueprints按钮，然后选择Open Level Blueprint，如图3-227所示。

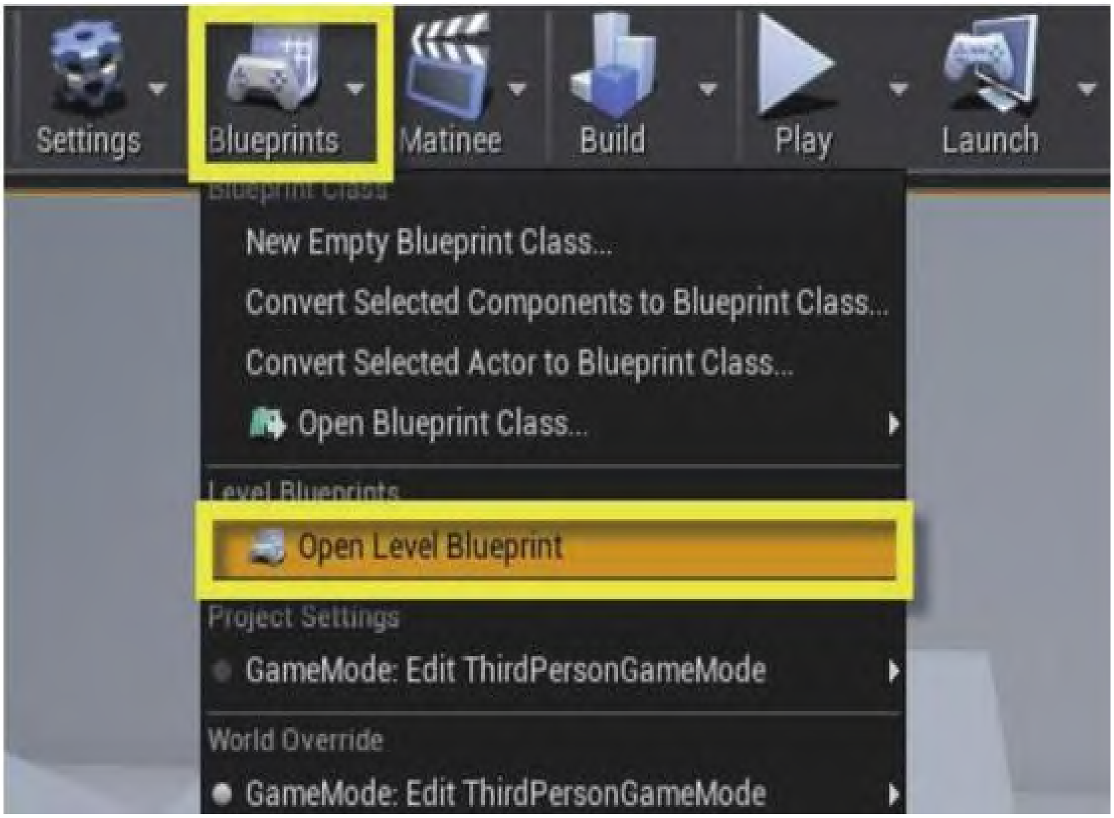


图3-227

**Step 9** 在图表中右击并选择Create a Reference to SM_Bush节点，如图3-228所示。

图3-228

此操作将在关卡中为SM_Bush创建一个引用。

如画面中不存在上图列出的节点，可在关卡中选择SM_Bush并重试。

**Step 10** 在图表中右击并添加Event Begin Play和Get Player Character节点。

**Step 11** 在Get Player Character的引出连线后搜索并添加Cast To ThirdPersonCharacter节点，如图3-229所示。

图3-229

这样我们便能访问ThirdPersonCharacter蓝图中的事件、函数、变量或属性。

**Step 12** 在As Third Person Character引脚的引出连线后搜索并使用Assign Start Fire节点，如图3-230所示。

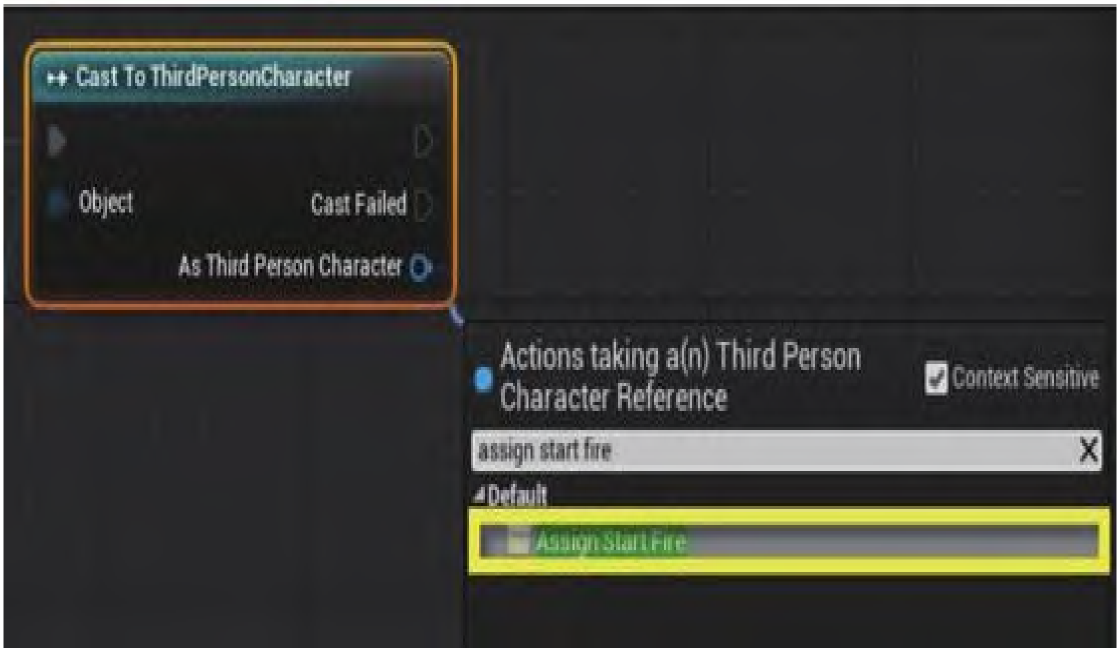



图3-230

此操作将新建两个节点：Bind Event和Custom Event。在角色蓝图中调用事件分配器时，Custom Event和与其相连的任意事件均会被执行（因它们与事件分配器绑定）。事件与事件分配器绑定的蓝图也将被执行。

**Step 13**　在Custom Event的引出连线后添加Spawn Actor from Class节点（Class设为Blueprint_Effect_Fire）。

**Step 14**　将SM_Bush的引出连线连接到Get Actor Transform，将其用作Spawn Actor节点的Spawn Transform，如图3-231所示。

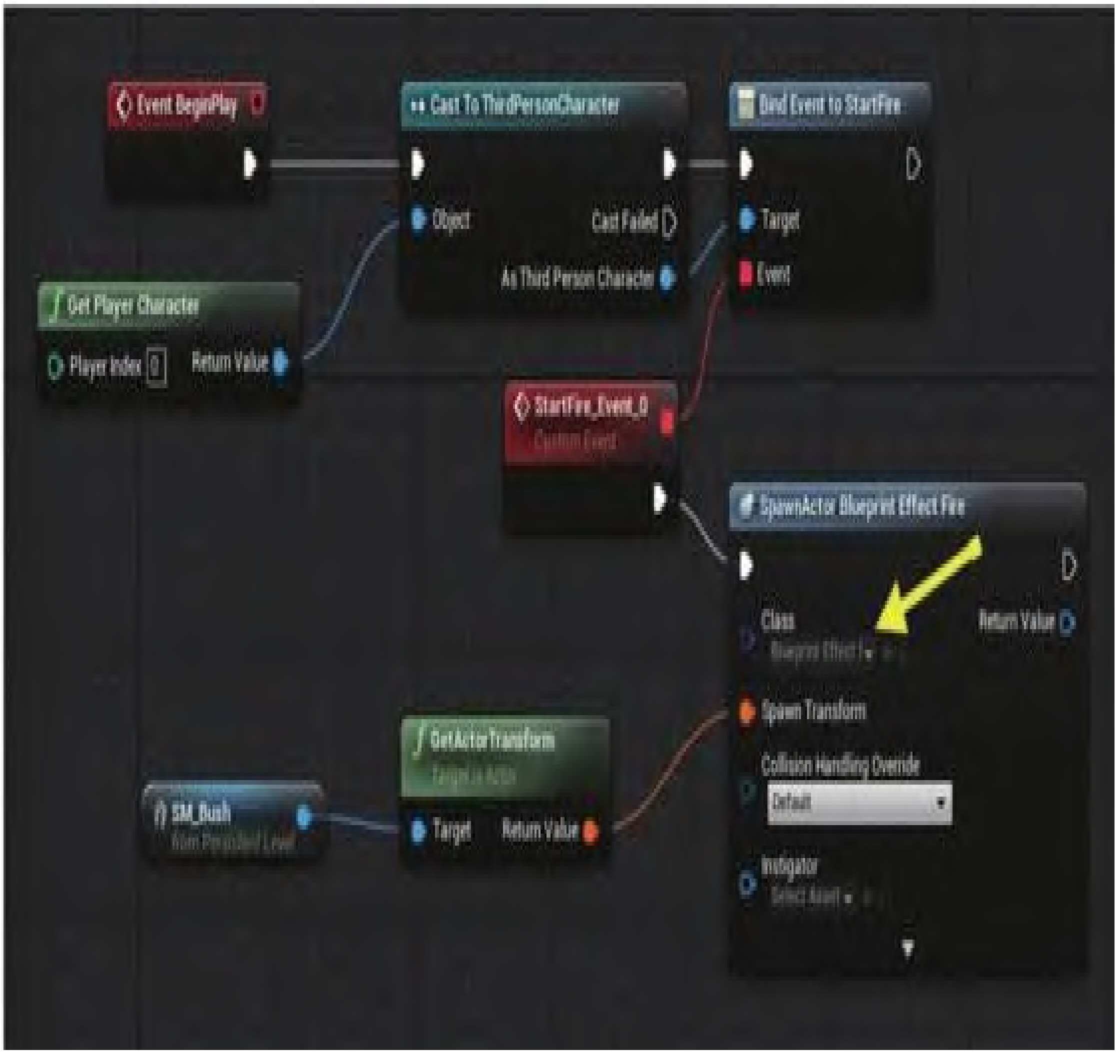



图3-231

所有节点连接完毕后，图表外观和上图相似。

**Step 15**　单击Compile按钮，关闭关卡蓝图，然后单击Play按钮在编辑器中开始游戏，如图3-232所示。

图3-232

按下F键后，火焰特效将出现在灌木丛中。

## 3.3.10　创建宏

从本质上来说，宏和节点的重叠图表原理相同。它们都有通道节点设计的进入点和离开点。每个通道可拥有任意数量的执行或数据引脚。在其他蓝图和图表上使用时，这些引脚在宏节点上均为可见。

在本例中，我们将创建一个宏，用于检查确定玩家角色是否拥有足够能量执行跳跃。如能量足够，将扣除相应能量值，显示当前数值到屏幕上，然后允许玩家执行跳跃。如能量不足，屏幕上将显示玩家能量不足，无法跳跃。

在此实例中，我们使用的是启用了新手内容的Blueprint Third Person项目。

**Step 1**　在Content/ThirdPersonBP/Blueprints文件夹中，打开ThirdPersonCharacter蓝图。

**Step 2**　在我的蓝图窗口中单击Add Macro按钮，如图3-233所示。

**Step 3**　将创建一个新的宏，选中后按F2键将其重命名为EnergyCheck，如图3-234所示。

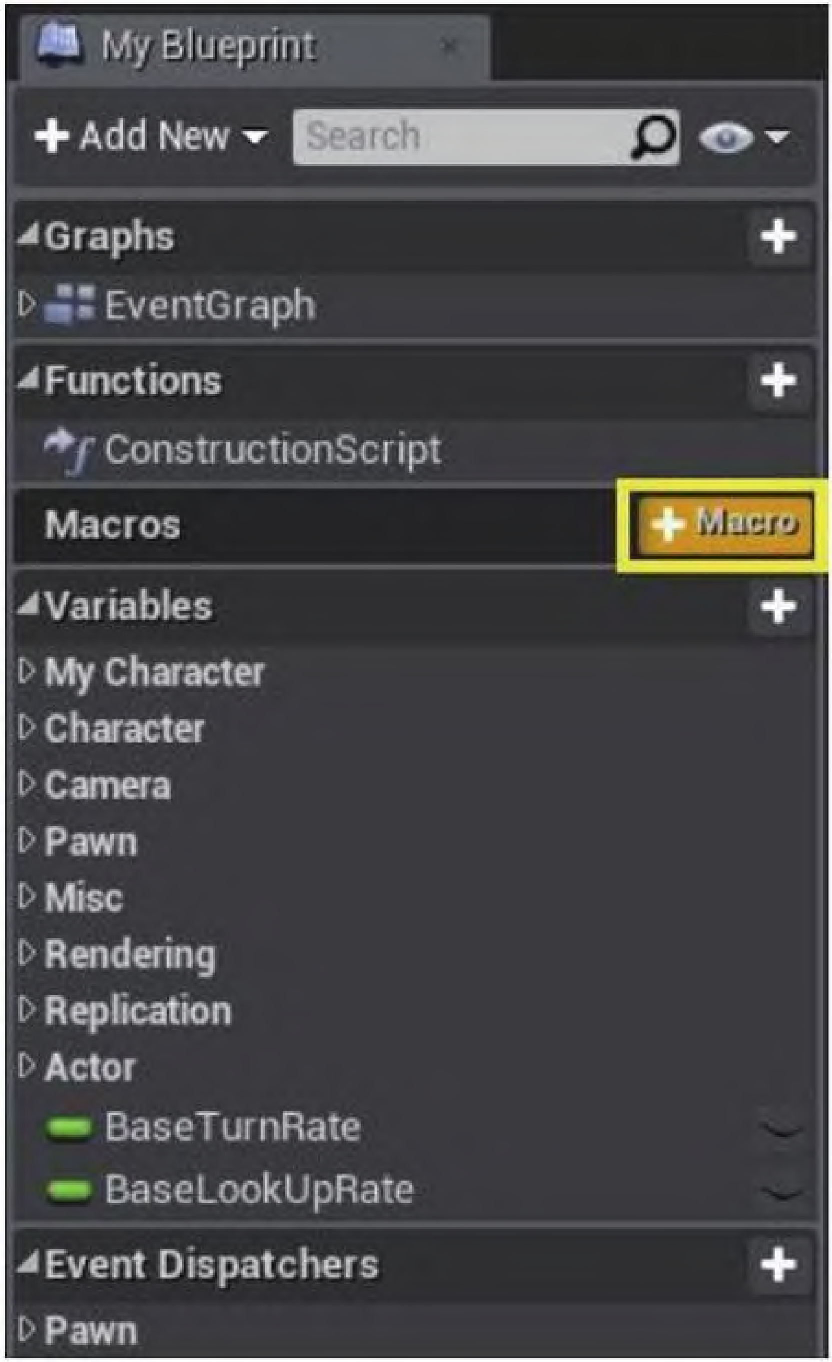

My Blueprint
Add New
Search
Graphs
EventGraph
Functions
ConstructionScript
Macros
Macro
Variables
My Character
Character
Camera
Pawn
Misc
Rendering
Replication
Actor
BaseTurnRate
BaseLookUpRate
Event Dispatchers
Pawn

图3-233

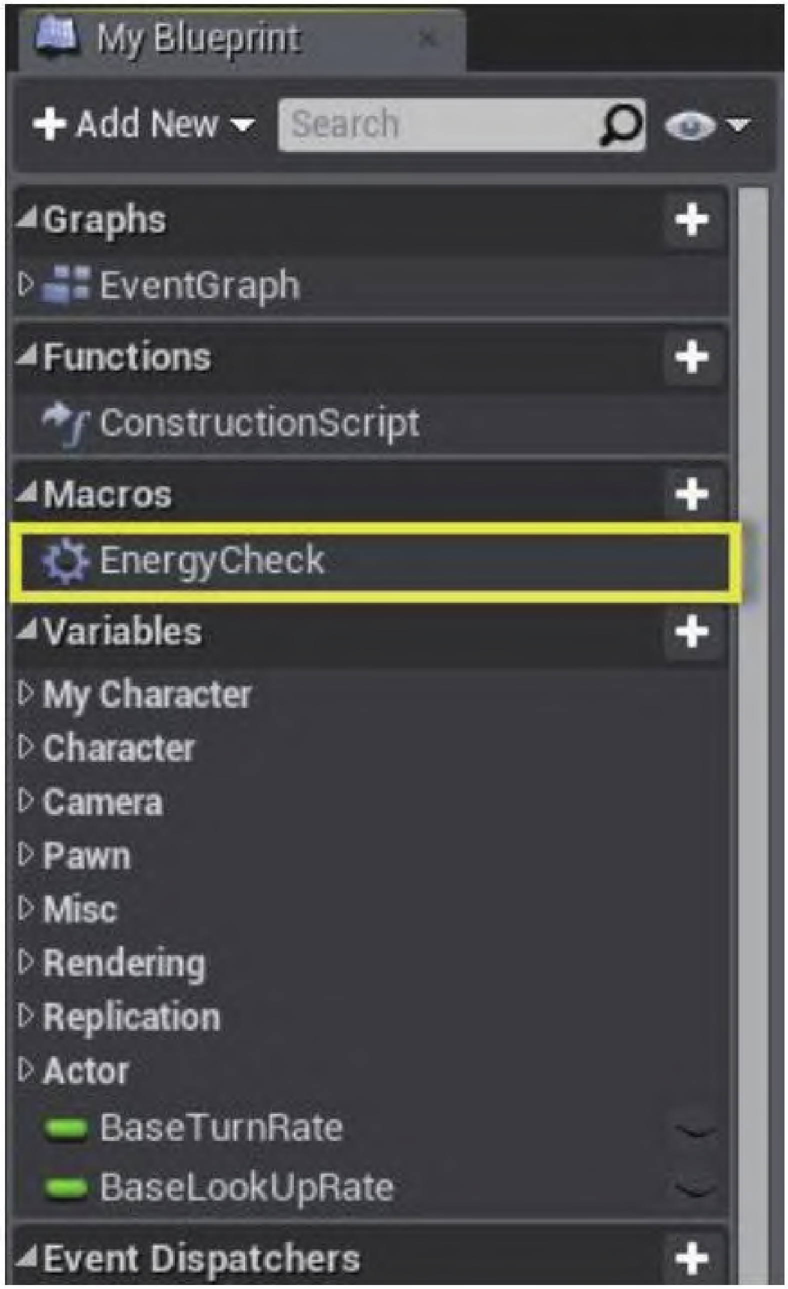



图3-234

**Step 4** 选中宏后，在Details面板中单击New按钮创建1输入和2输出，如图3-235所示。

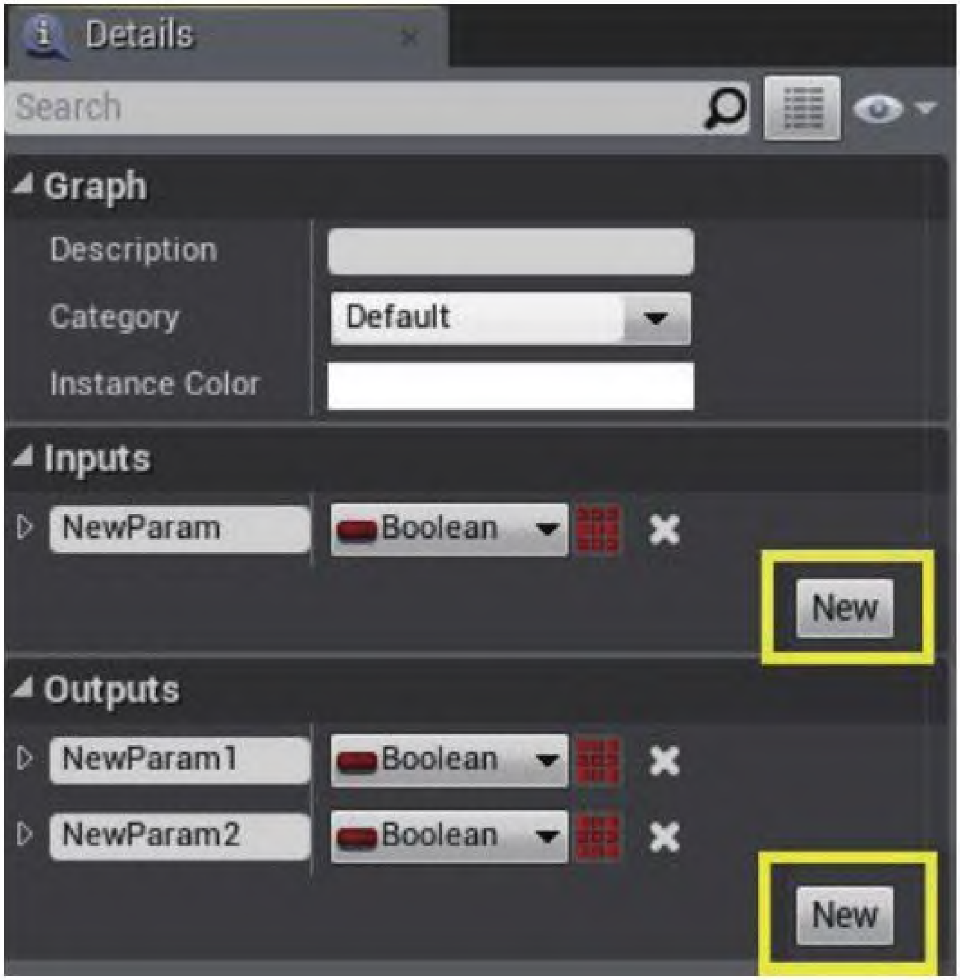



图3-235

此操作将在宏节点上创建输入/输出节点，可用于向宏传输数据（或从宏传输数据）。

**Step 5**　在文本框中为输入节点设置名称BeginCheck，然后将类型改为Exec（可执行引脚）。

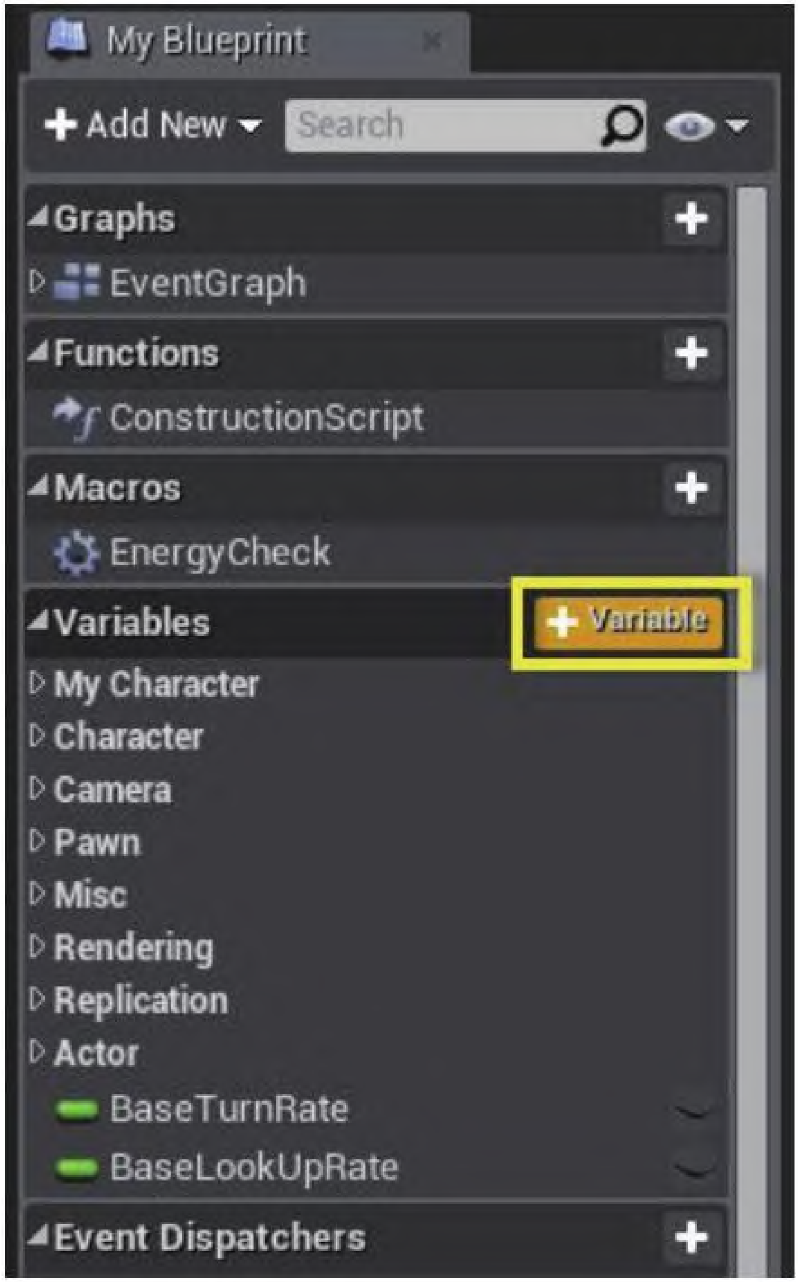
**Step 6**　将两个输出节点类型设为Exec，并将一个命名为HasEnergy，另一个命名为NoEnergy，如图3-236所示。

图3-236

在输入节点上，使用一个名为BeginCheck的执行引脚开始宏。将创建脚本，用以检查玩家是否拥有足够能量执行跳跃（如是，则将其与HasEnergy引脚连接起来）。如玩家无足够能量，则将其与NoEnergy引脚连接起来。执行宏时，HasEnergy引脚将填充Jump指令，而NoEnergy引脚则不会。

**Step 7**　在My Blueprint窗口中单击Add Variable按钮，如图3-237所示。

**Step 8**　将变量命名为Energy并将其类型改为Float，单击工具栏上的Compile按钮，将值设为100，如图3-238所示。



My Blueprint
Add New
Search
Graphs
EventGraph
Functions
ConstructionScript
Macros
EnergyCheck
Variables
Variable
My Character
Character
Camera
Pawn
Misc
Rendering
Replication
Actor
BaseTurnRate
BaseLookUpRate
Event Dispatchers


图3-237

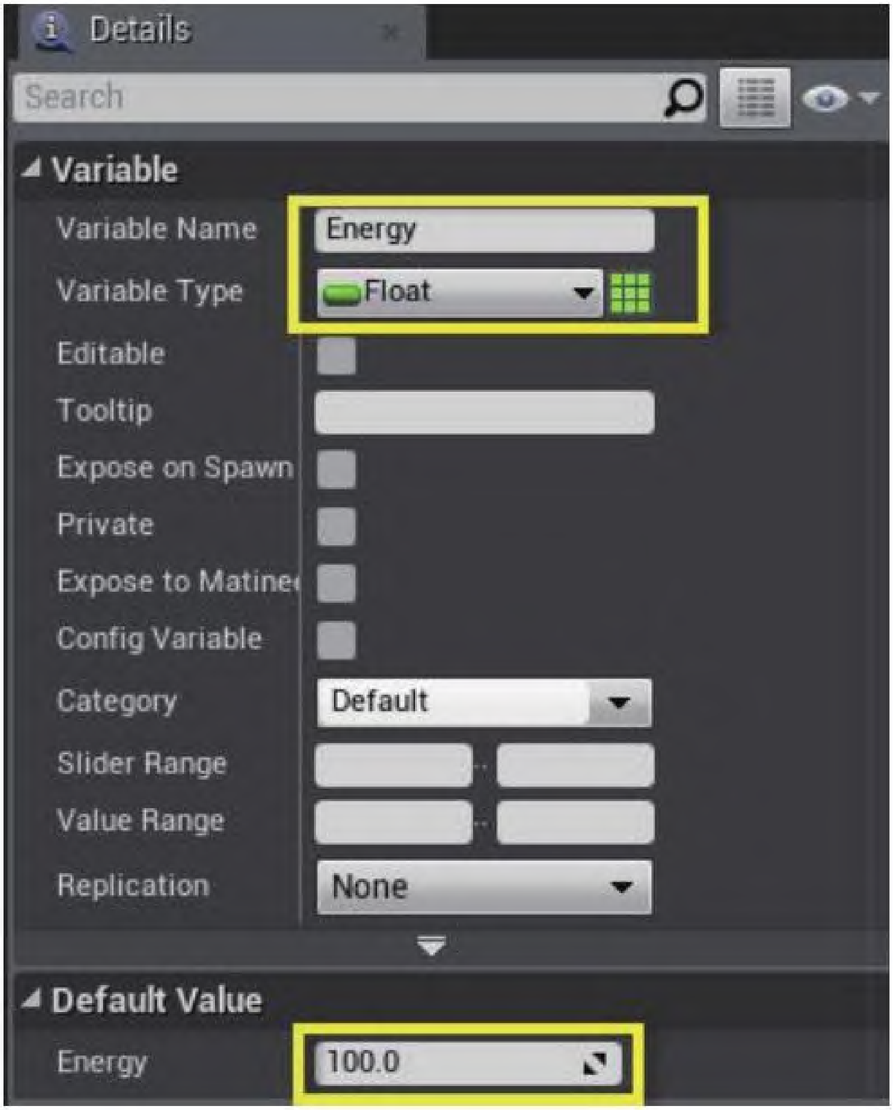



图3-238

现在便为玩家的能量（初始值为100）设置了一个变量。

**Step 9**　在Inputs旁的图表中，按住B键并单击创建一个Branch节点。

**Step 10**　按住Ctrl键从My Blueprint窗口将Energy拖入图表并连接到一个>节点，然后再连接到Branch，如图3-239所示。

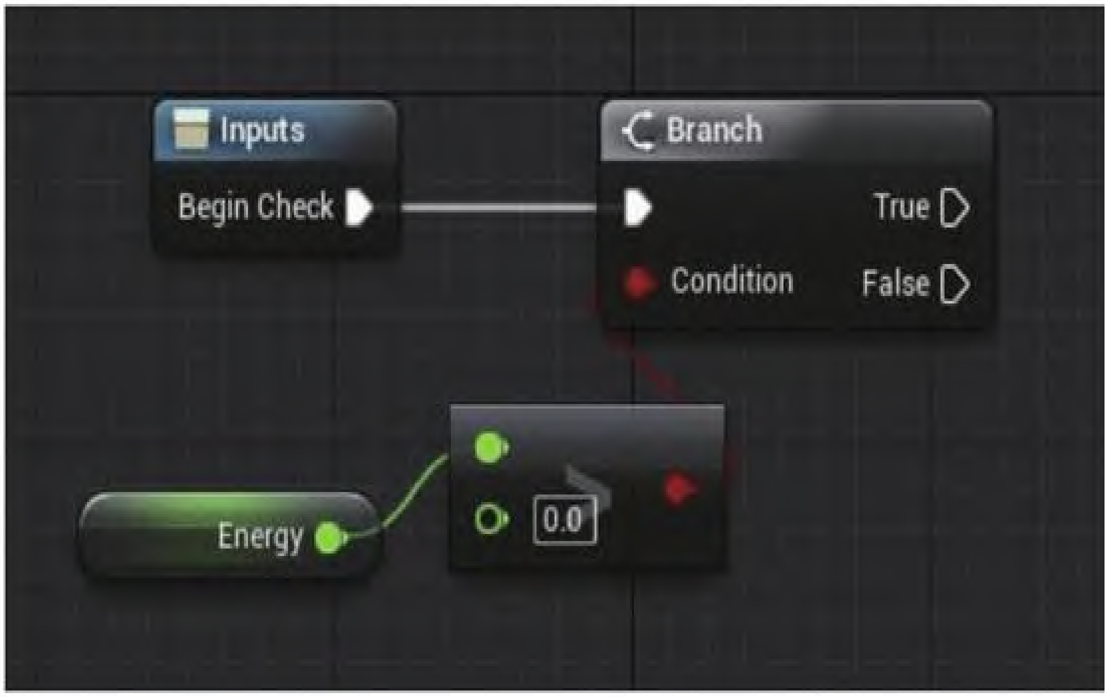



图3-239

在此我们使用Branch确定能量是否大于零。如是，执行True ；如否，执行False。

**Step 11**　按住Alt键拖入Energy变量添加一个Set节点。

**Step 12**　在此按住Ctrl键拖入Energy并连接到一个节点，将值设为10，然后插入Set节点，如图3-240所示。

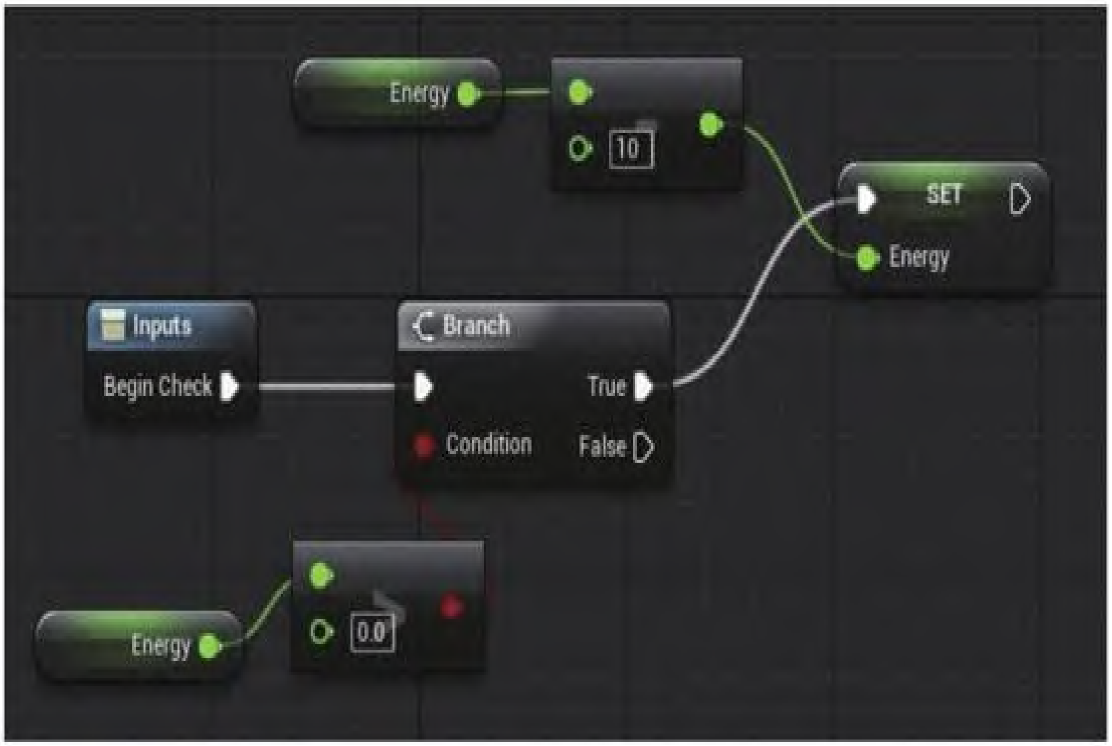



图3-240

该脚本意味着能量大于零时，将从当前能量值中减去10。

**Step 13**　在图表中右击添加一个Print String节点，然后将Energy和In String引脚连接起来，如图3-241所示。

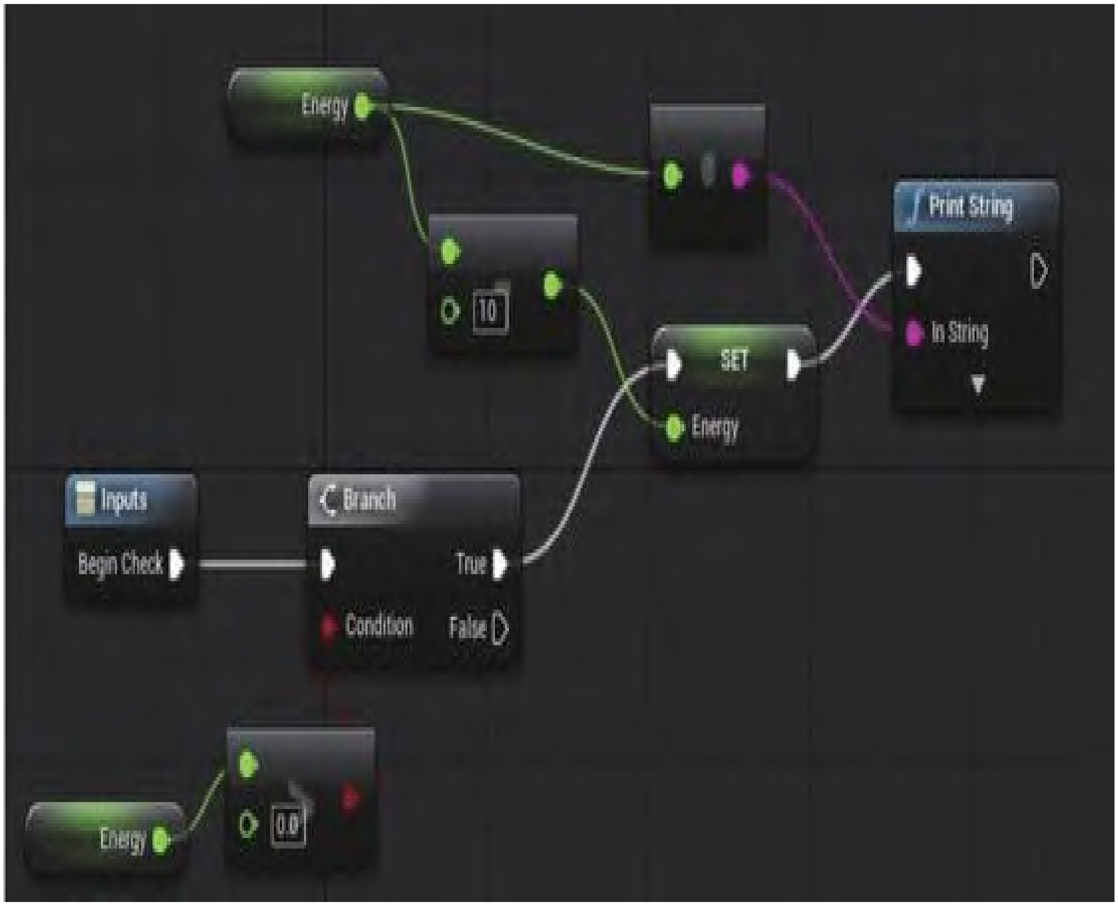


图3-241

添加一个转换节点，它将把能量值转换为数值显示在屏幕上。

**Step 14** 关闭Branch节点的False引脚，添加一个Print String并在框中输入文本“Out of Energy！”。

**Step 15** 将两个Print String节点与HasEnergy和NoEnergy引脚连接起来，如图3-242所示。

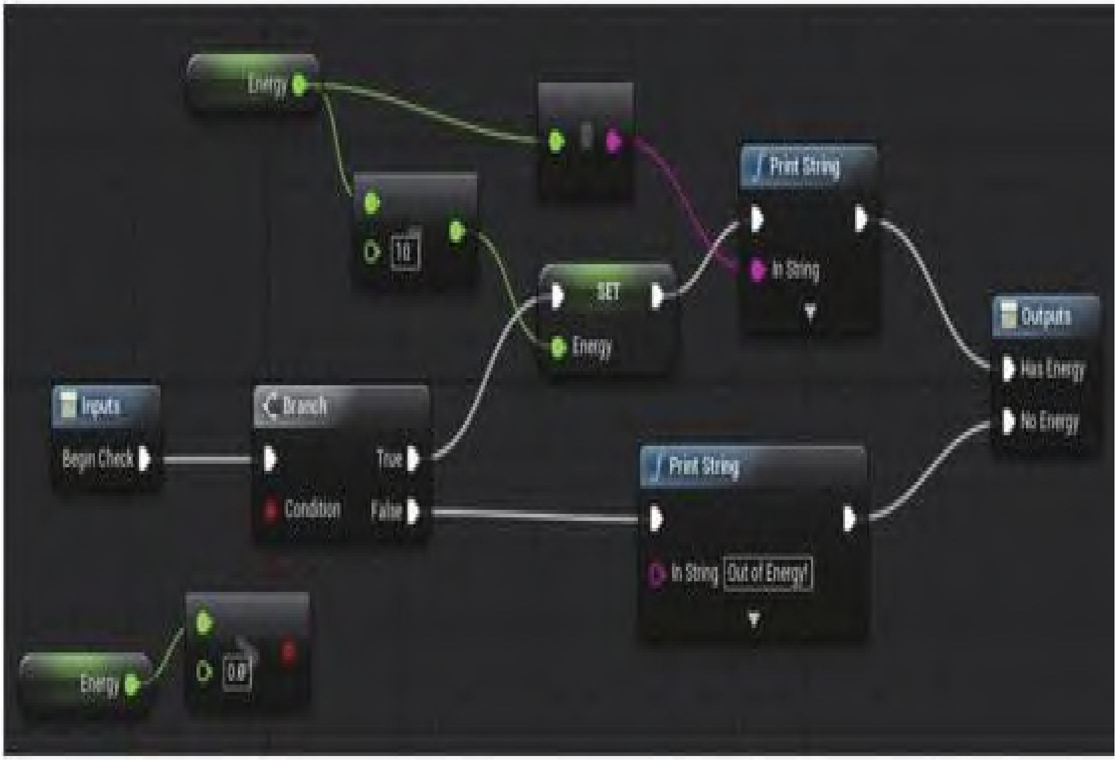



图3-242

现在宏的设置为检查Energy变量（如适用，则从该变量减去数值），确定玩家是否拥有足够能量，进而决定玩家是否可以执行跳跃。现在，我们需要在按下跳跃键后跳跃动作被执行之前应用宏。

**Step 16** 在事件图表上，拖动InputAction Jump节点的Pressed引脚并搜索Energy Check，如图3-243所示。

图3-243

可以看到创建的宏列于Utilities下，宏图标旁是它的名称。

**Step 17** 宏被添加后，Jump脚本的外观与图3-244所示相似。

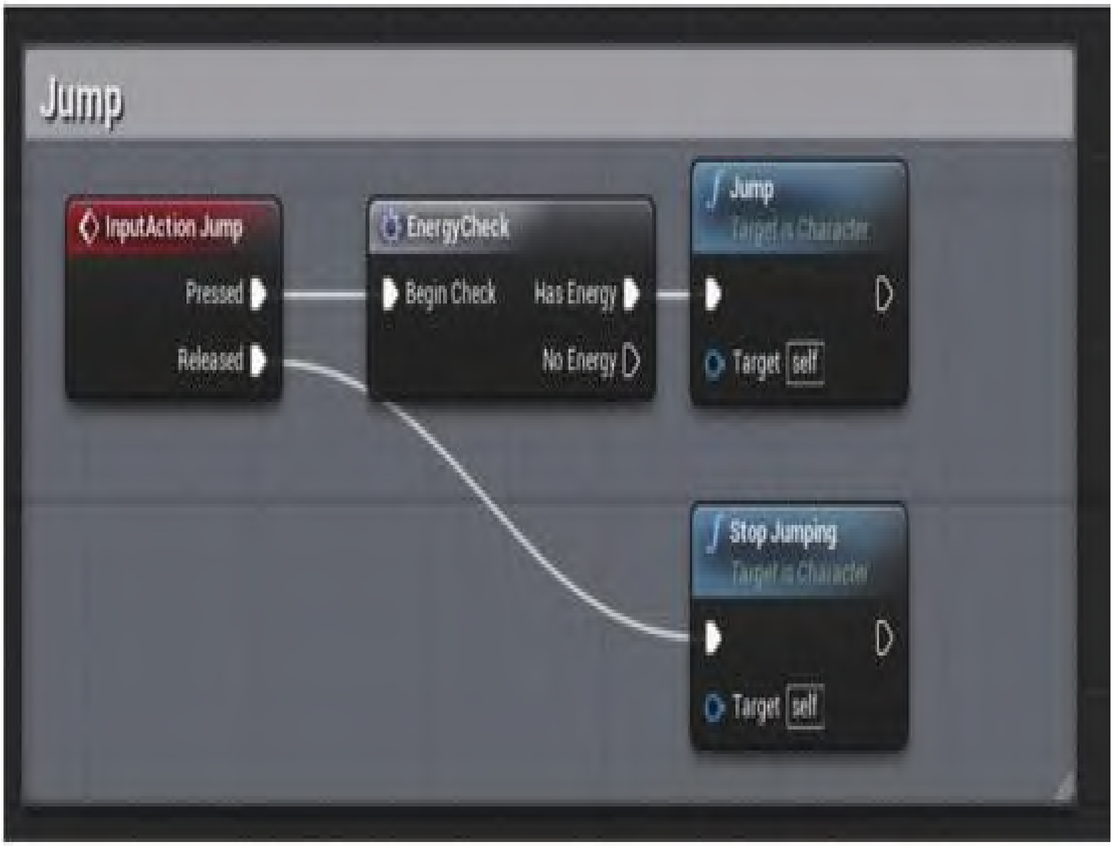



图3-244

**Step 18** 单击Compile和Save按钮，然后关闭蓝图。

**Step 19** 在主工具栏中单击Play按钮即可在编辑器中进行游戏。

按下空格键执行跳跃时，在屏幕左上角可看到Energy的值。当Energy为零时，无法执行跳跃。

这是使用宏将脚本执行并合并为一个单独节点、改善事件图表和主要角色脚本易读性的基本范例。除此之外还可在其他情况下调用这个宏：存在其他消耗玩家能量的动作，需要确定玩家是否有能量执行此操作（如攻击）。按下攻击键，然后在HasEnergy执行引脚后执行攻击操作，再运行这个宏确认玩家是否有足够的能量执行攻击。

## 3.3.11 放置节点

### 3.3.11.1 拖放

第一种方法是拖放，从MyBlueprint窗口将节点拖放到图表上。举例而言，在下图中可将PlayerHealth变量拖入几个变量中并放置在图表上，以便在脚本中使用，如图3-245所示。

单击将一个变量拖进图表窗口后出现两个选项：Get和Set。选择Get将创建一个Getter节点，可获取变量或变量值；而选择Set将创建一个Setter节点，可对变量值进行设置。根据节点的类型，可能出现只有Get一个选项的情况，如图3-246所示。

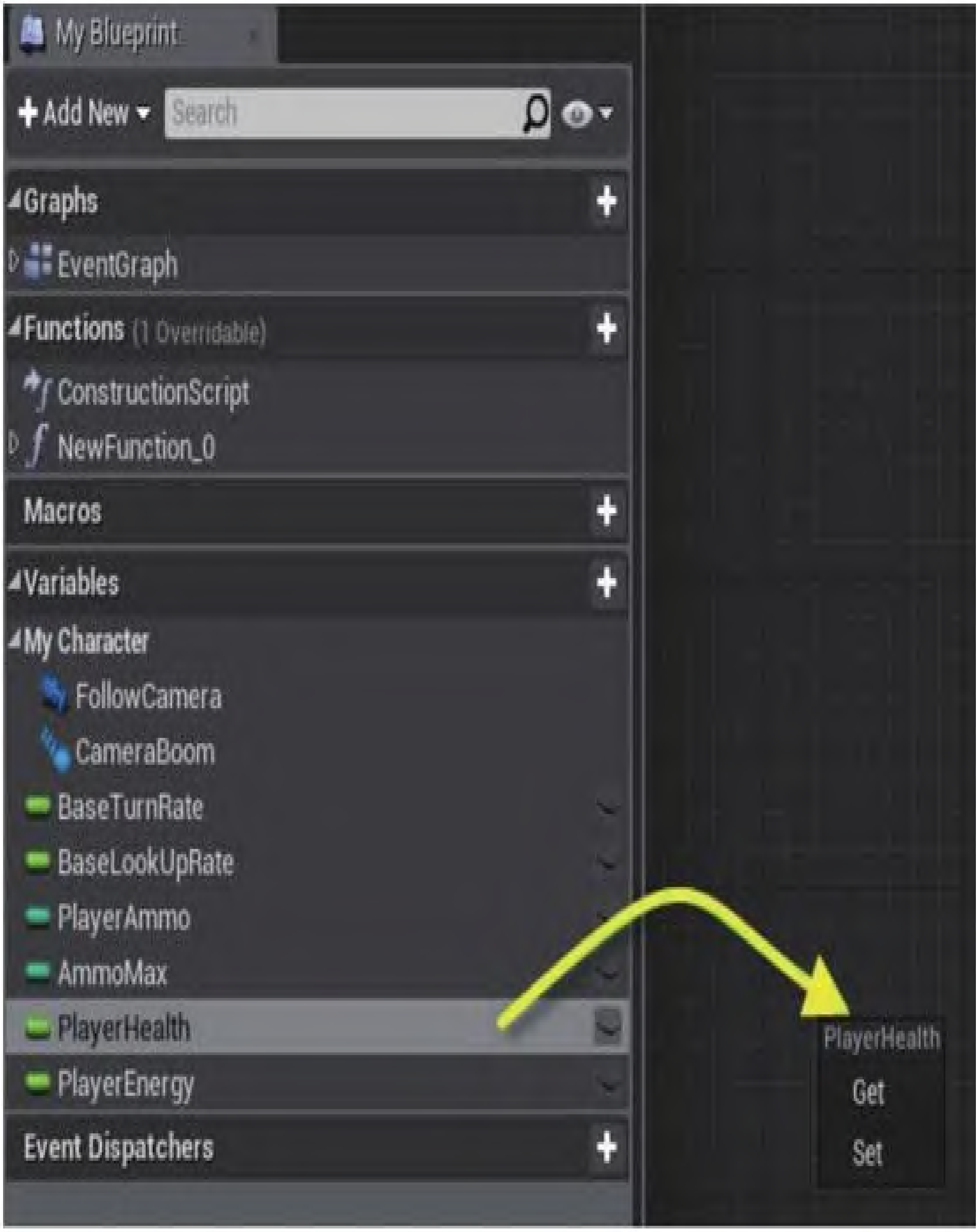

My Blueprint
Add New
Search
Graphs
EventGraph
Functions (1 Overridable)
ConstructionScript
NewFunction_0
Macros
Variables
My Character
FollowCamera
CameraBoom
BaseTurnRate
BaseLookUpRate
PlayerAmmo
AmmoMax
PlayerHealth
PlayerEnergy
Event Dispatchers
PlayerHealth
Get
Set

图3-245

图3-246

在上图中，上方节点是Getter，下方节点是Setter。举例而言，Getter可用作游戏元素的条件（如获取游戏角色体力值，确认数值是否高于特定范围，如否，判定游戏角色死亡）。借用相同例子从另一方面来说，Setter可用于增加玩家的体力值：为玩家体力值变量设置数值。

同理可将创建好的函数和宏拖入图表，如图3-247所示。

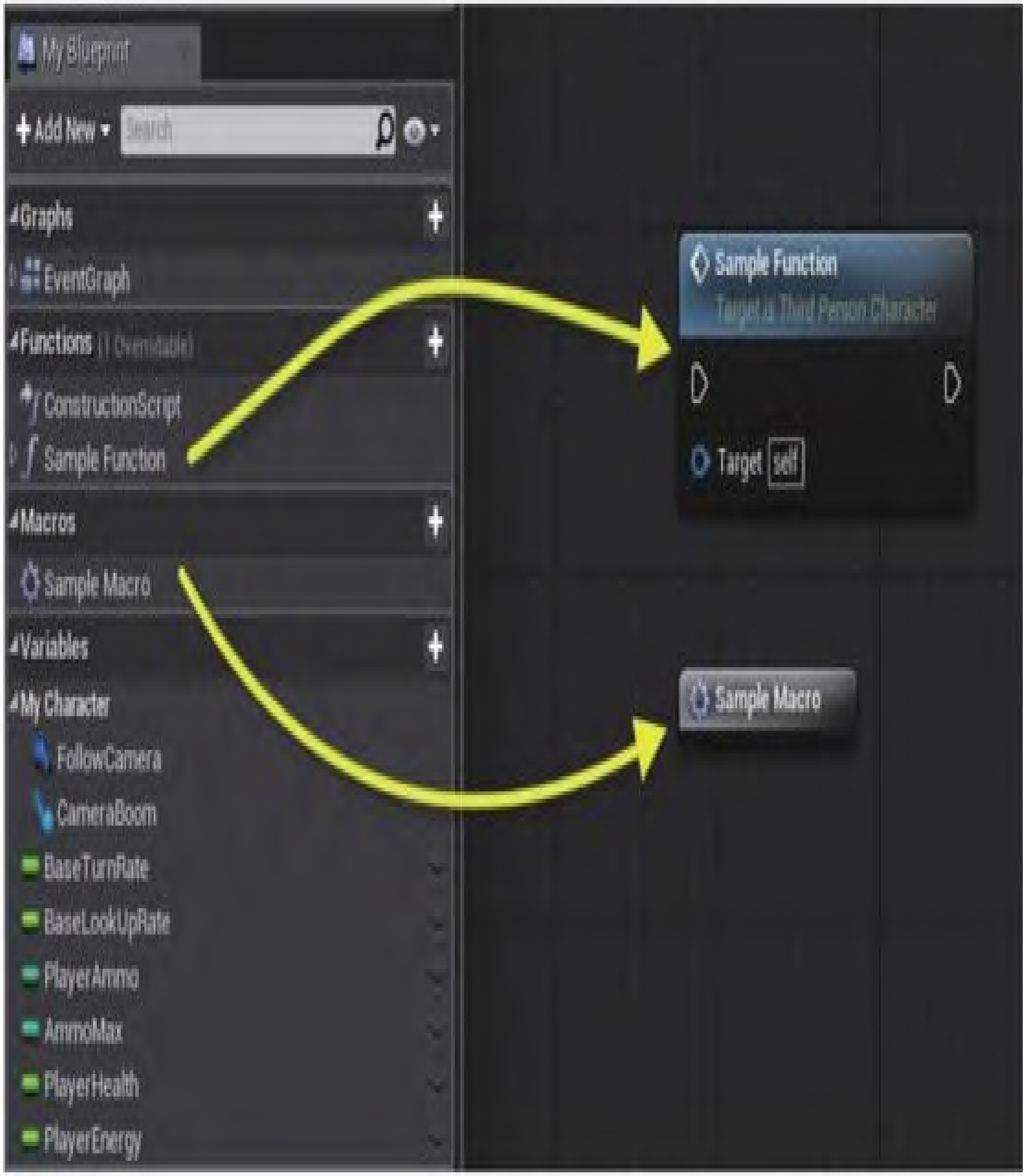



图3-247

拖入一个Event Dispatcher后，一些特殊上下文操作将变为可用状态，如图3-248所示。

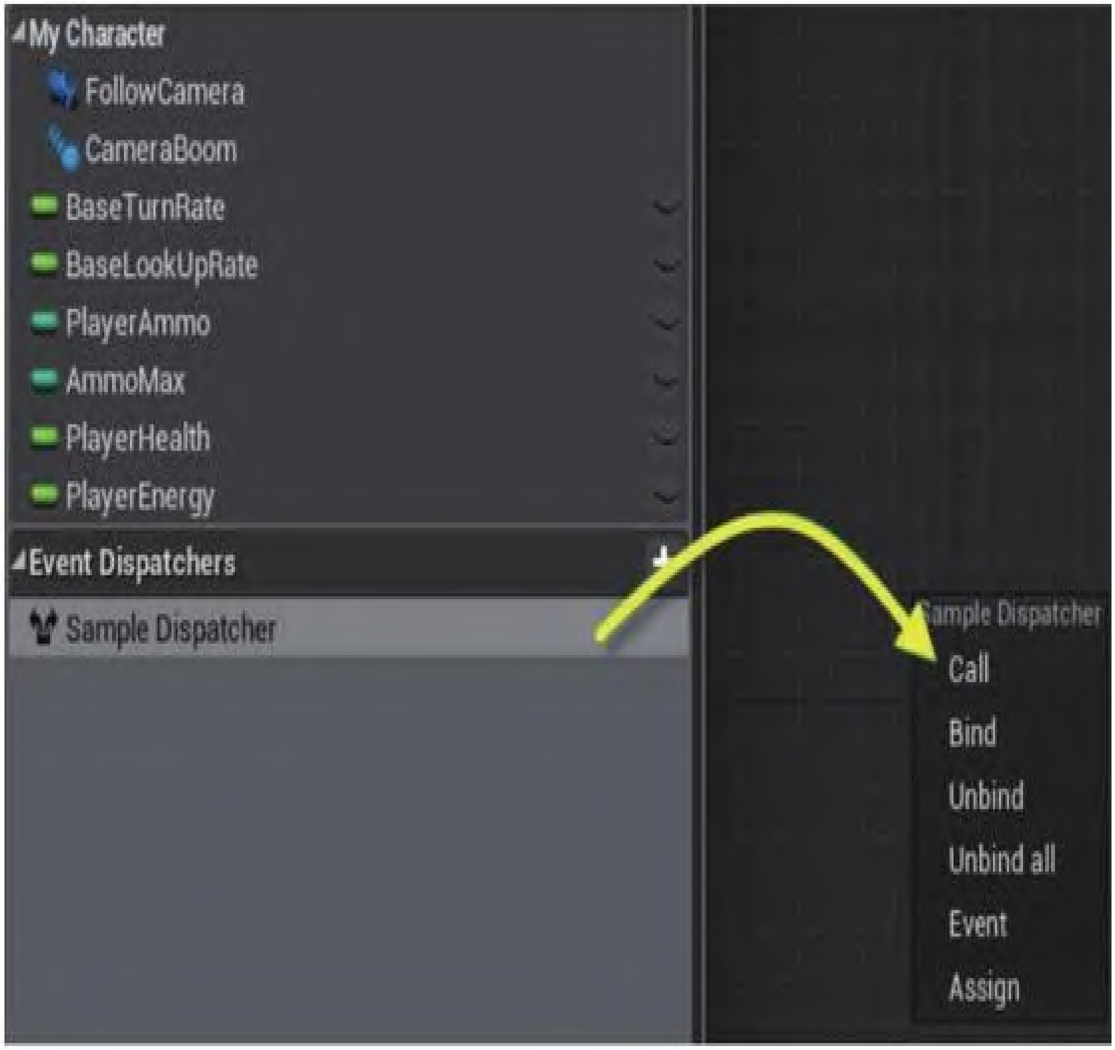



图3-248

快捷菜单在拖入Event Dispatcher时出现，可从中选择需要执行的操作。

#### 3.3.11.2　快捷菜单搜索

多数情况下，可在蓝图图表中右击访问快捷菜单放置节点，如图3-249所示。

图3-249

从上图菜单中展开任意类目（或子类目），然后选择需要的节点添加至图表中。

窗口右上角有一个名为Context Sensitive的选项。它为默认开启，禁用此选项后将基于当前上下文自动筛选菜单中显示的选项，如图3-250所示。

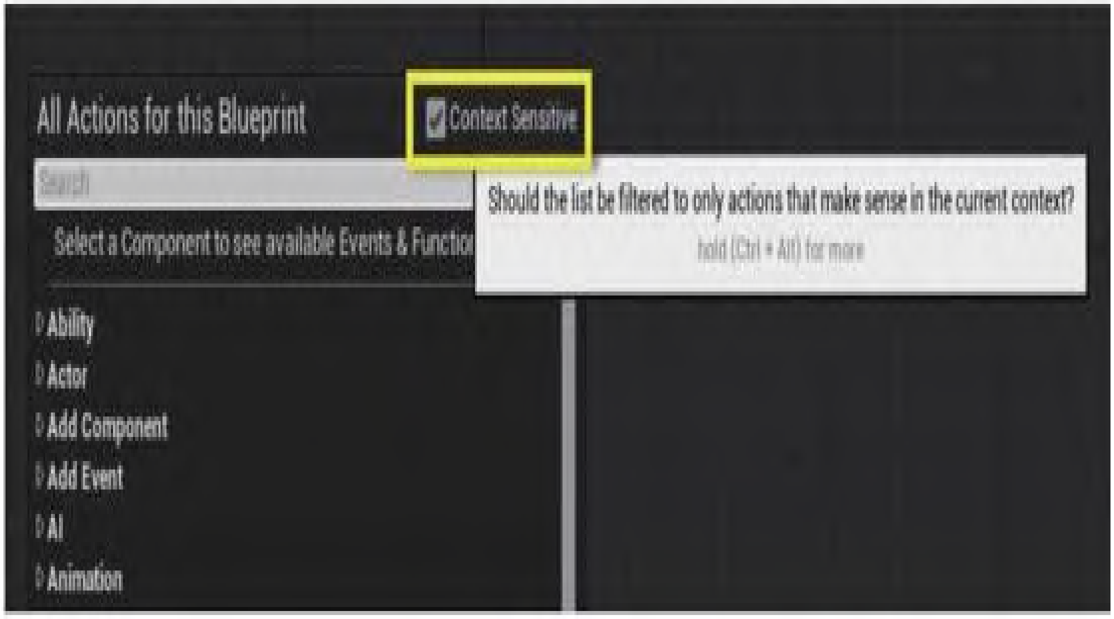



图3-250

Context Sensitive选项开启时右击并搜索Animation，便会出现筛选列表，如图3-251所示。

然后，如取消勾选Context Sensitive并搜索Animation，便会出现所有与animation相关的内容，如图3-252所示。

All Actions for this Blueprint
Context Sensitive
Animation
Animation
Get Current Montage
Is Playing Networked Root Motion Montage
Is Playing Root Motion
Lock AIResources with Animation
Play Anim Montage
Stop Anim Montage
Unlock AIResources with Animation
Components
Skeletal Mesh
Get Animation Mode (Mesh)
Play Animation (Mesh)
Set Animation (Mesh)
Set Animation Mode (Mesh)

图3-251

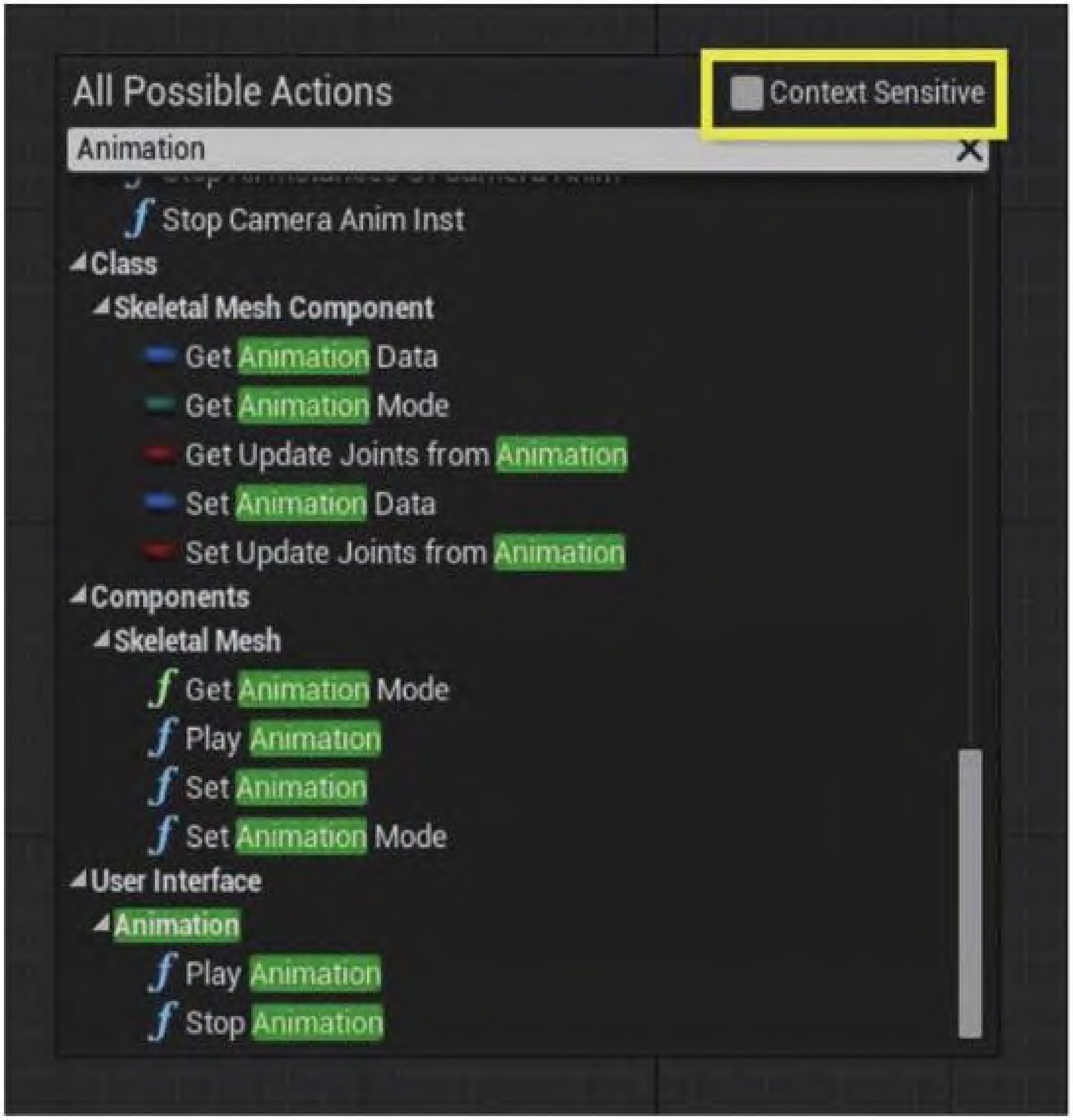



图3-252

右击可呼出快捷菜单，也可拖动现有节点访问快捷菜单，如图3-253所示。

图3-253

在上图中有一个Character Movement组件引用，拖动其输出引脚可添加连接上下文的节点。这些节点和被拖动的节点为相关，如图3-254所示。

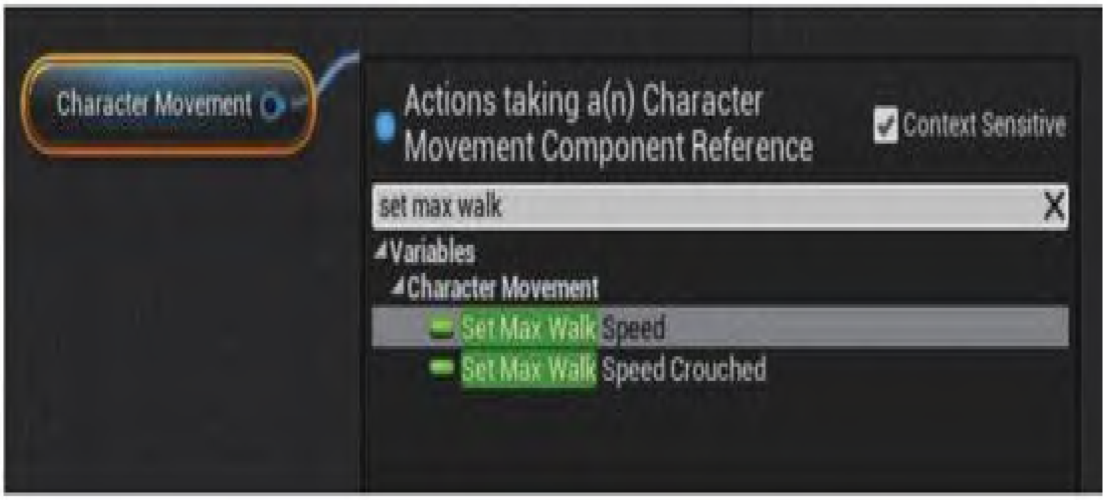



图3-254

在上图中，可搜索Set Max Walk，然后从菜单中选择Set Max Walk Speed对角色的最高步行速度进行设置。

#### 3.3.11.3　快捷键

也可使用快捷键替代（和使用）节点，提升工作效率。

### 3.3.12　使用阵列

如你不熟悉编程术语，理解阵列的最佳方式是将其看作包含在单一实体中的一批同类型变量。举例而言，如要创建一个Float阵列，该阵列只能包含Float变量。创建阵列后，可在阵列中访问个体变量并对其进行修改，或在阵列中一次性修改所有物品（在处理重复性工作时可极大提高工作效率）。

在本节中，你将学习如何在蓝图中创建阵列变量，以及如何使用Make Array节点创建阵列。理解如何创建阵列后，可在范例阵列用法章节查看使用阵列的范例。

#### 3.3.12.1　阵列变量

可通过下列步骤创建阵列变量：

**Step 1**　在蓝图中单击Add Variable按钮，为变量命名并设置变量类型，如图3-255所示。

在上图中创建了一个名为TextArray的文本变量。

**Step 2**　在变量的Details面板中，单击Variable Type旁的阵列网格按钮，如图3-256所示。

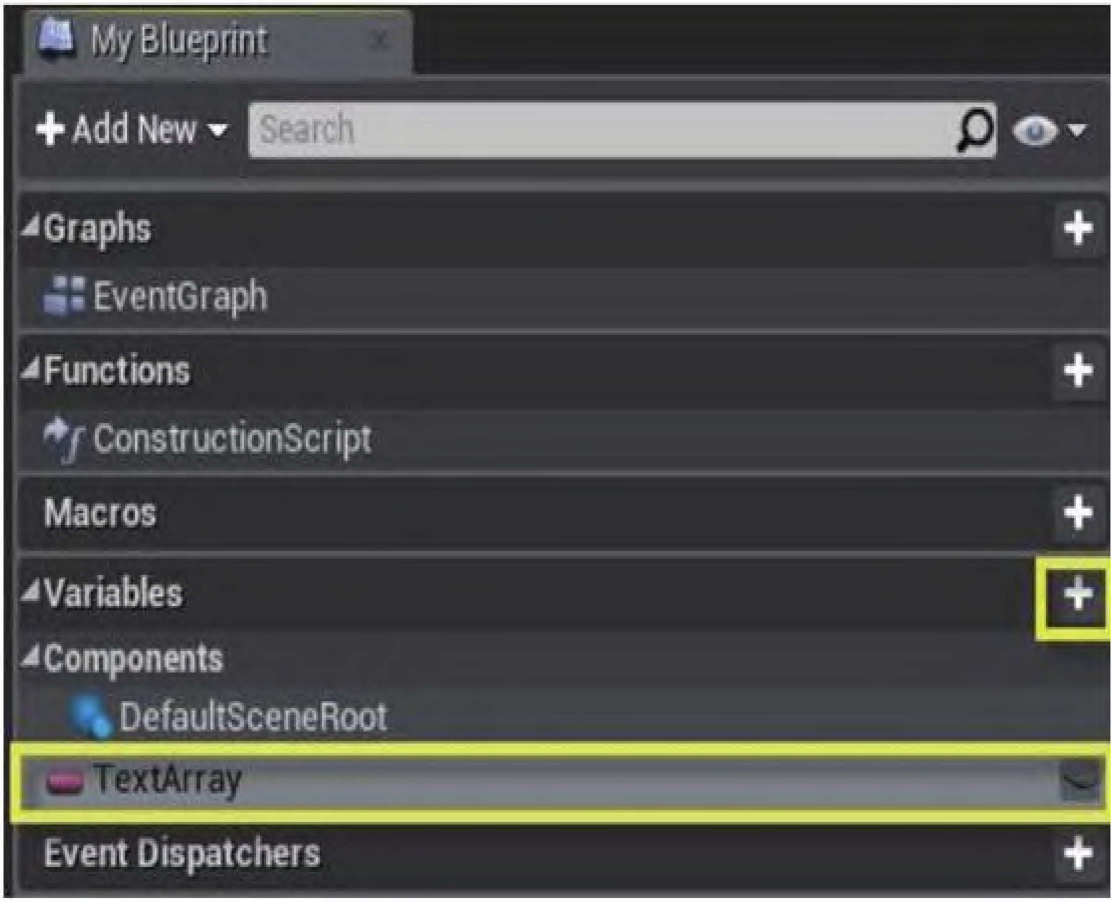



图3-255

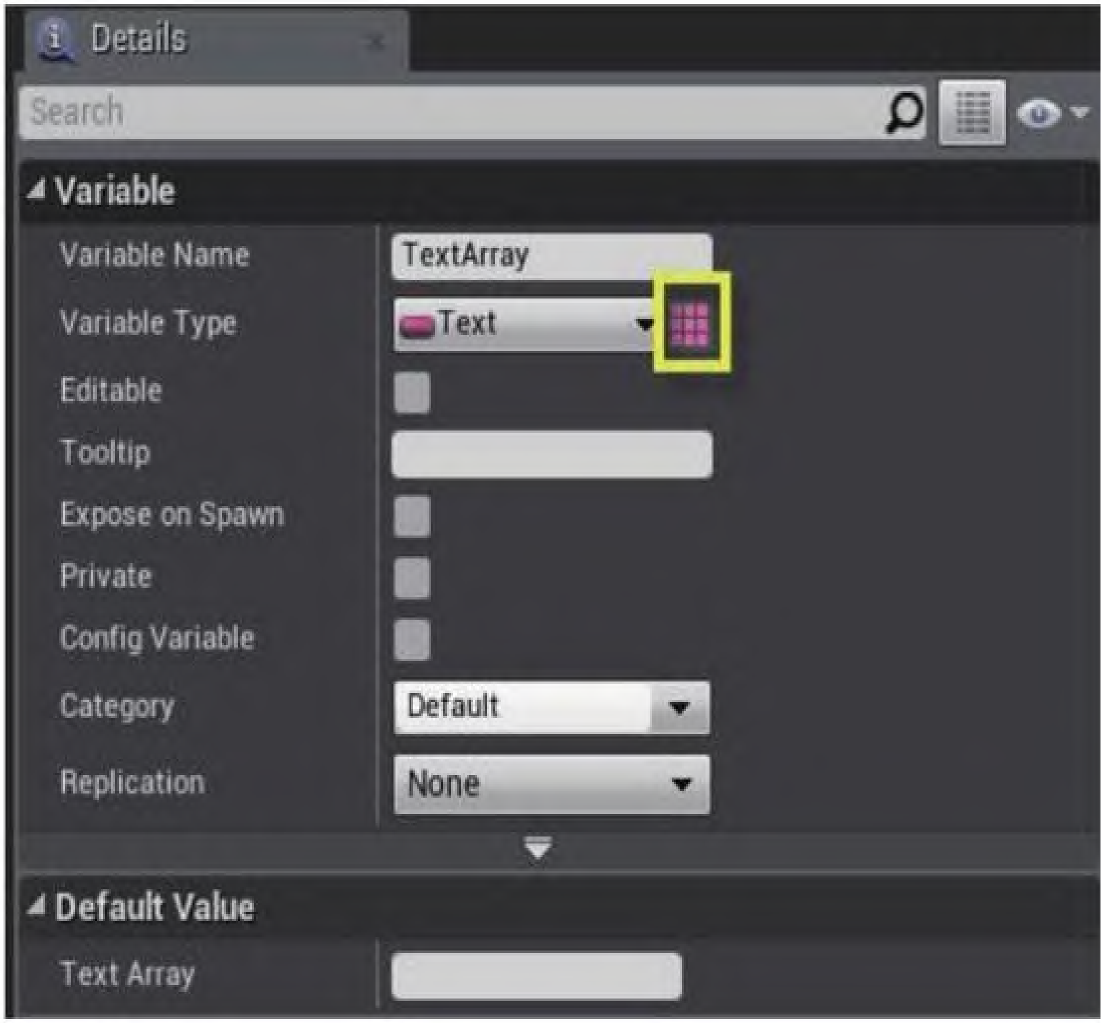



图3-256

此操作将把变量转换为阵列，如图3-257所示。

**Step 3** 单击Compile按钮，然后在Default Value下单击垃圾筐图标旁的+符号，为阵列添加内容，如图3-258所示。

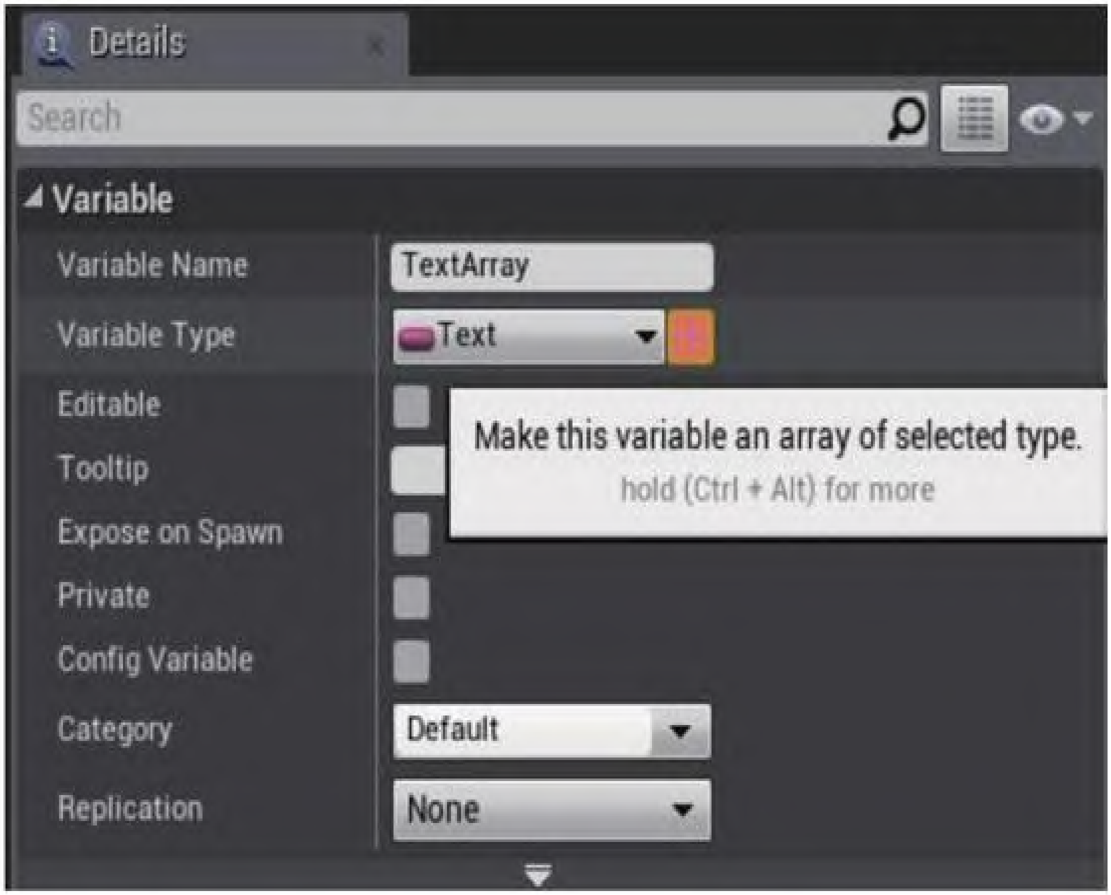

Details
Search
Variable
Variable Name
TextArray
Variable Type
Text
Editable
Make this variable an array of selected type.
hold (Ctrl + Alt) for more
Tooltip
Expose on Spawn
Private
Config Variable
Category
Default
Replication
None

图3-257

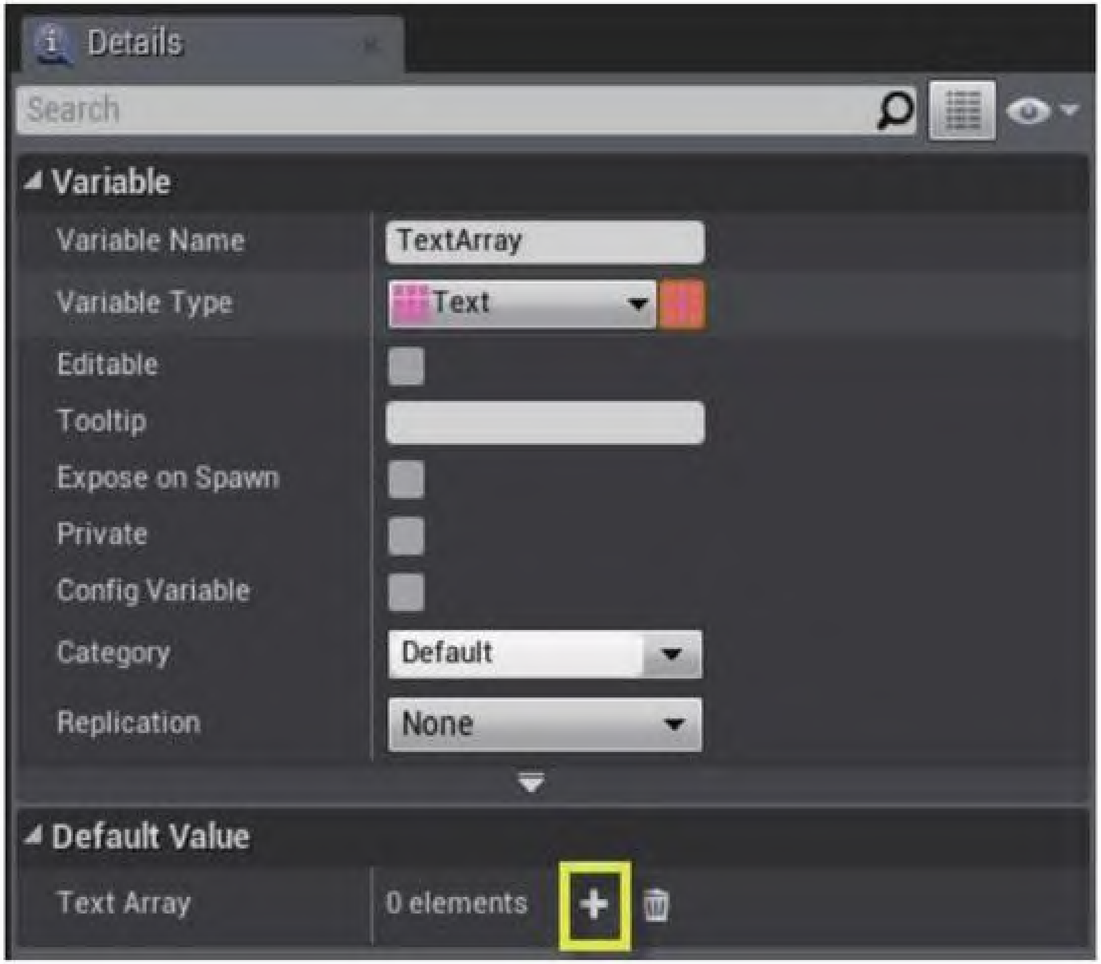



图3-258

**Step 4** 可添加任意数量的元素到阵列，并在索引中指派其数值，如图3-259所示。

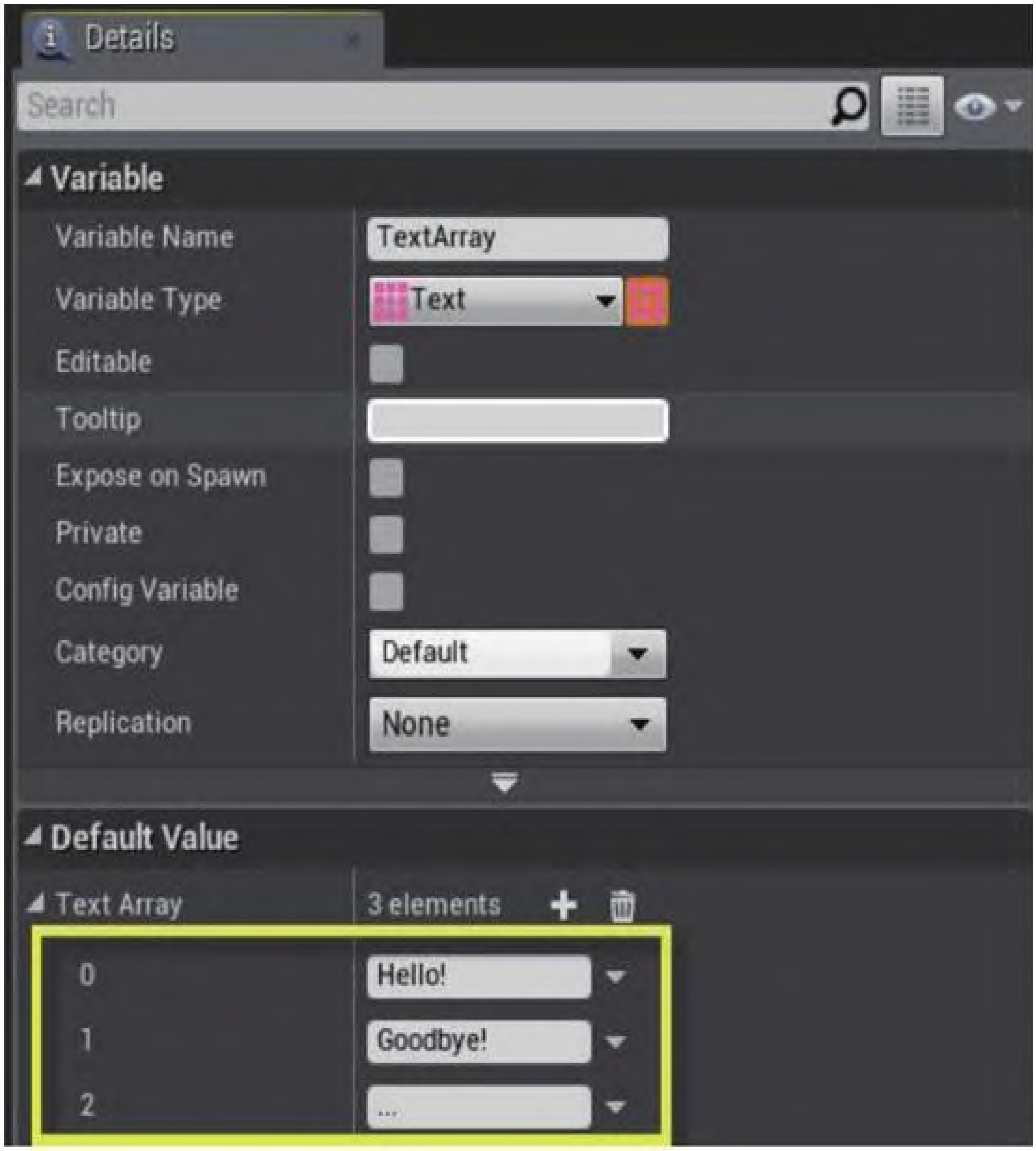



图3-259

在上例中我们创建了一个文字阵列，首个输入设为Hello！，第二个输入GoodBye！，第三个输入设为……。该例可用于用NPC，基于设置的各种条件以不同文本回应玩家。

如上图所示，阵列使用从零开始的索引值，意味着首个索引为0而非1。填充阵列时请牢记这点。在通过索引值访问阵列中的内容时，如需从第一个元素开始访问，需要输入的索引值为0而非1。如需访问第二个元素，则需输入索引值1，以此类推。

### 3.3.12.2 创建阵列

创建阵列变量时，通过变量的默认值设置定义出现在阵列中的内容。存在阵列动态填充的情况，举例而言，可使用关卡中存在的Actors填充阵列。在这种情况下，可使用Make Array节点以一系列内容创建一个阵列，无须创建阵列变量并指派默认值。

举例而言，关卡中需要摄像机Actors阵列，使其在每个Actor间循环。

以下步骤说明如何使用Make Array节点。

**Step 1** 在关卡中放置数个Actor（在本例中为CameraActors）后，将它们全部选中并在Level Blueprint中打开。

**Step 2** 在图表中右击并选择Add Reference to Selected Actors选项。

**Step 3** 为阵列所含的Actor添加引用后，在图表中右击并添加Make Array节点，如图3-260所示。

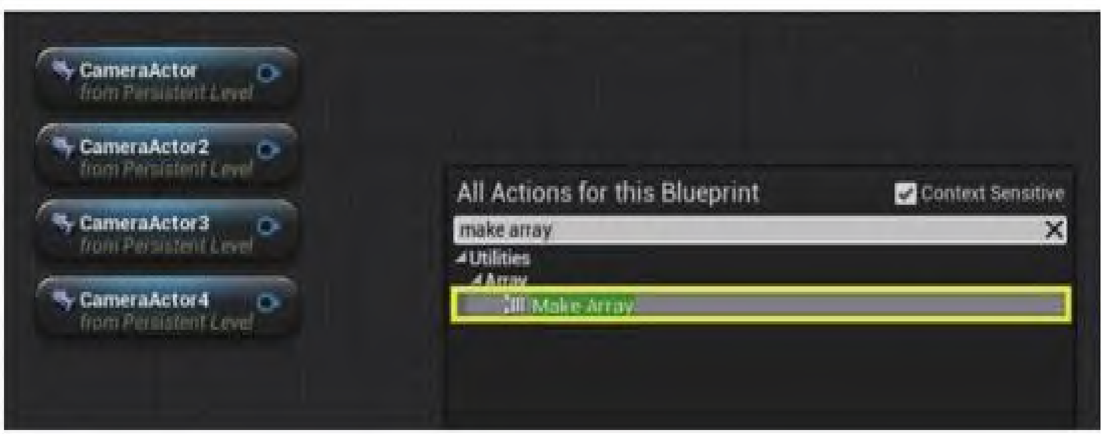



图3-260

**Step 4** 在Make Array节点上，单击+符号添加每个Actor出现在阵列中所需的引脚数量，如图3-261所示。

**Step 5** 拖动一个Actor并将它和Make Array节点上的0（或任意其他引脚）连接起来，如图3-262所示。

图3-261

图3-262

此操作将把阵列换转为所选Actor的阵列类型（Error报错提示将在编译后消失）。

**Step 6** 将其他Actor和Make Array节点上其余的引脚连接起来，如图3-263所示。

图3-263

Step 7　拖动Array引脚，搜索并添加Get节点，如图3-264所示。

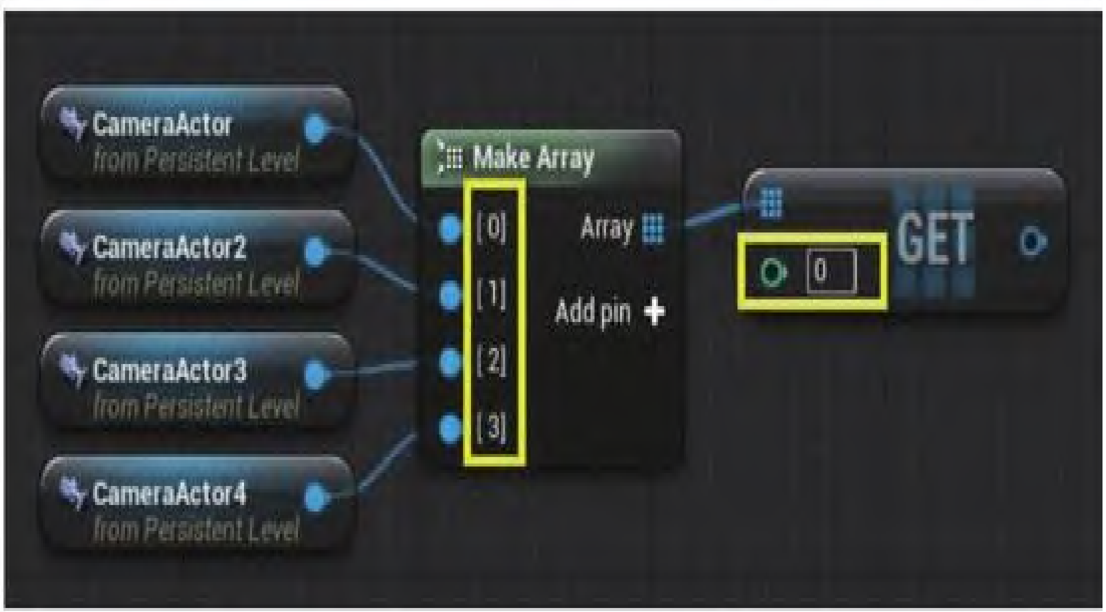



图3-264

Get节点上框中的值是从Make Array节点访问（Get节点）的索引（阵列元素）。

它也可能是下图中名为Get Camera的整数变量，如图3-265所示。

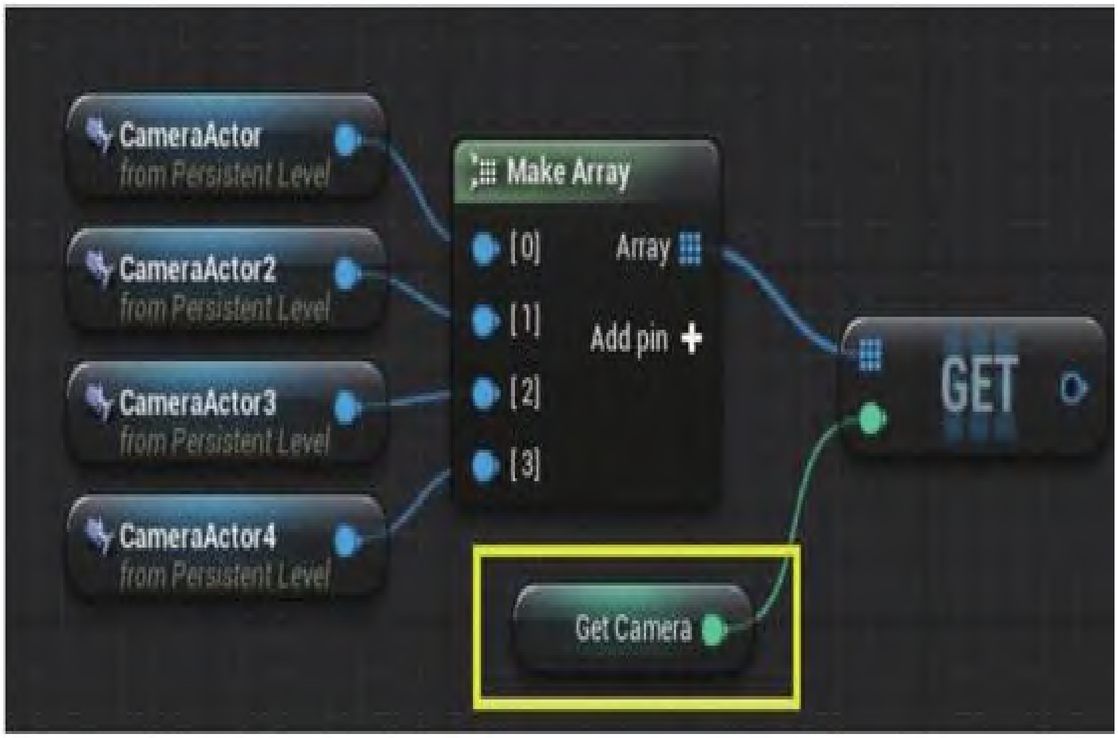



图3-265

利用一些脚本形态，可将Get Camera设为整数值，有效地改变Get节点中选择的Camera Actor。下图中的设置可用于在关卡中放置的摄像机Actors之间产生循环，如图3-266所示。

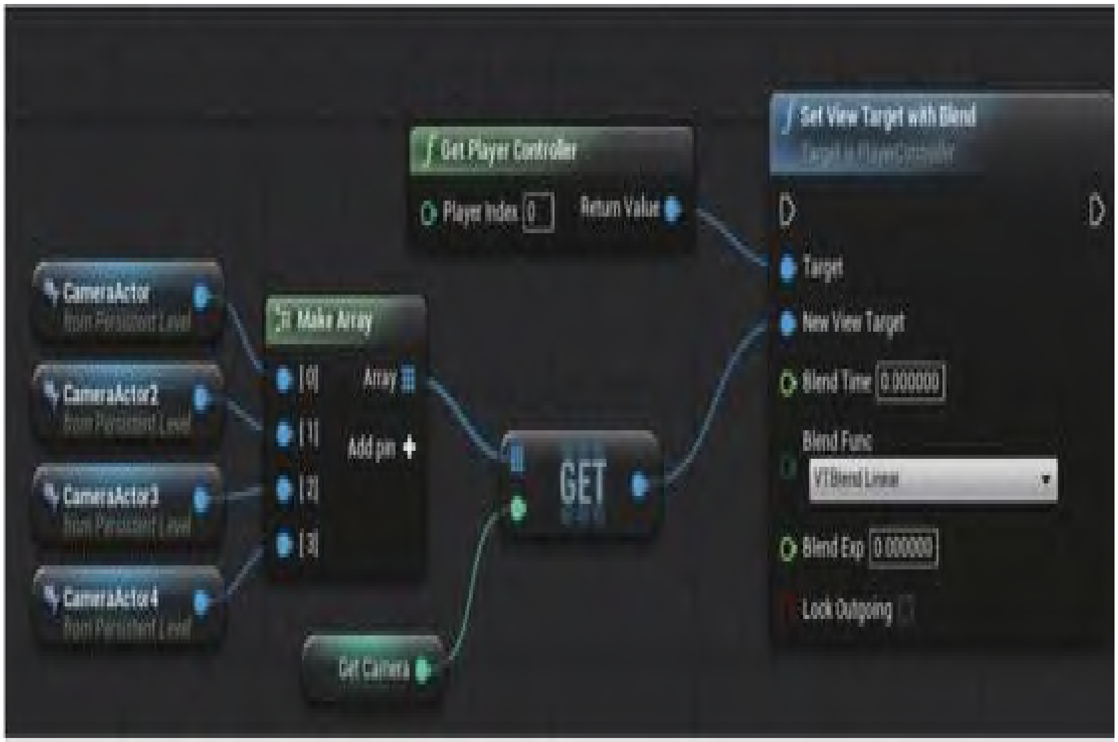



图3-266

### 3.3.12.3 范例阵列使用

在本例中，我们将使用阵列在关卡中的随机位置生成随机蓝图效果。

在此实例中，我们使用的是启用新手内容的Blueprint Third Person模板项目。

**Step 1** 在项目中Basic标签上的Modes菜单中，将5个Target Points拖入关卡，如图3-267所示。

图3-267

它们将作为效果生成的地点。

**Step 2**　在World Outliner中，搜索Target，然后按住Shift按键选择全部目标点，如图3-268所示。

**Step 3**　在主工具栏上，单击Blueprints按钮，然后单击Open Level Blueprint按钮，如图3-269所示。

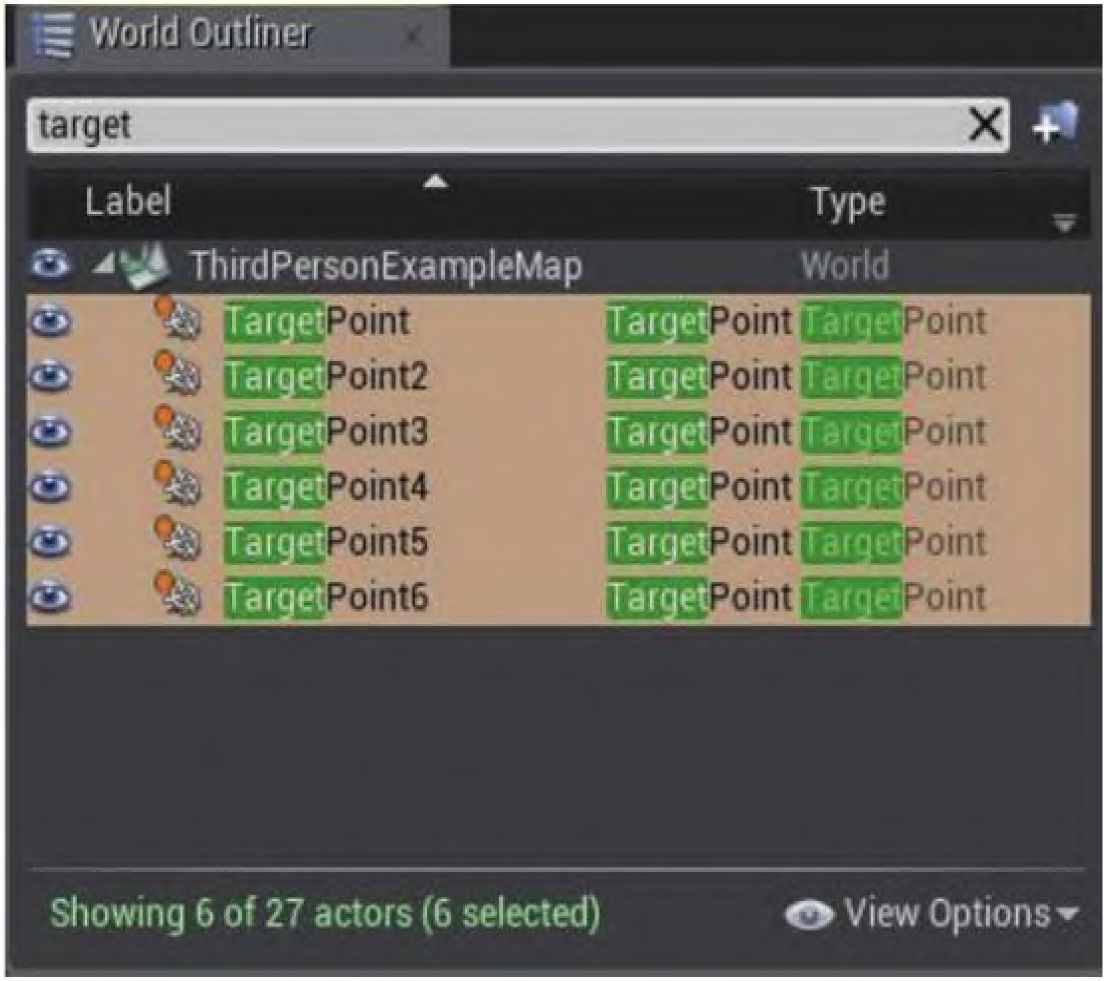

World Outliner
target
Label
Type
ThirdPersonExampleMap
World
TargetPoint
TargetPoint
TargetPoint
TargetPoint2
TargetPoint
TargetPoint
TargetPoint3
TargetPoint
TargetPoint
TargetPoint4
TargetPoint
TargetPoint
TargetPoint5
TargetPoint
TargetPoint
TargetPoint6
TargetPoint
TargetPoint
Showing 6 of 27 actors (6 selected)
View Options

图3-268

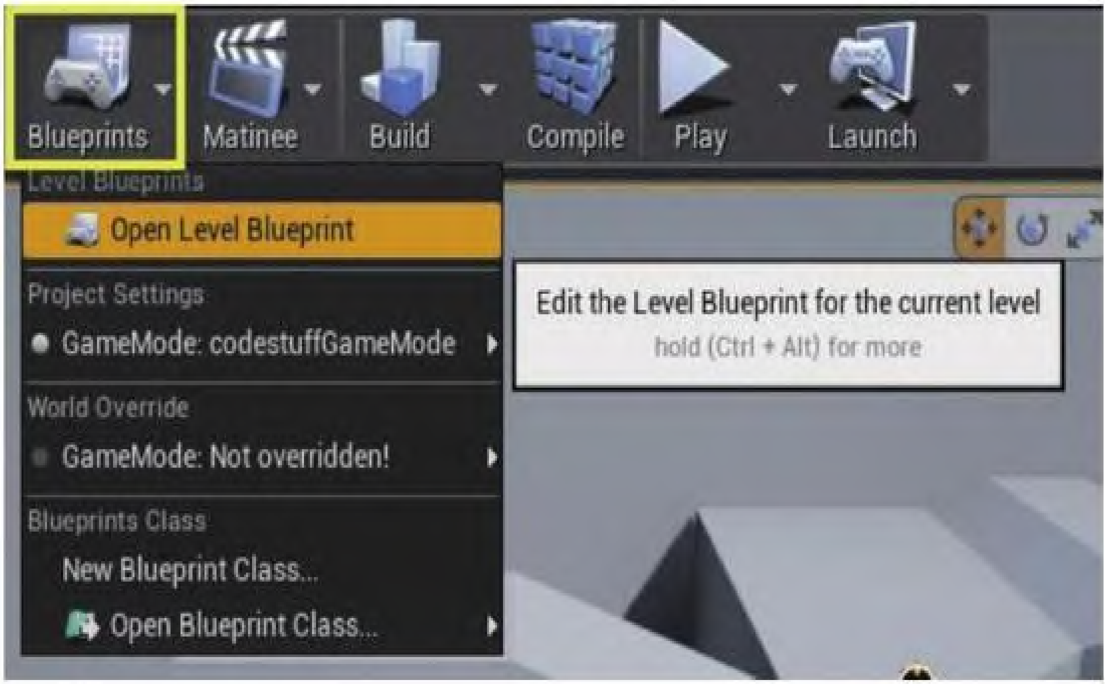



图3-269

**Step 4**　在图表中右击并选择Create References to 5 selected Actors选项，如图3-270所示。

此操作将为关卡中所有目标点创建引用，以便进行访问。

**Step 5**　再次在图表中右击，搜索并添加Make Array节点，如图3-271所示。

图3-270

图3-271

**Step 6**　在Make Array节点上单击四次+符号，将引脚添加至节点，然后连接目标点，如图3-272所示。

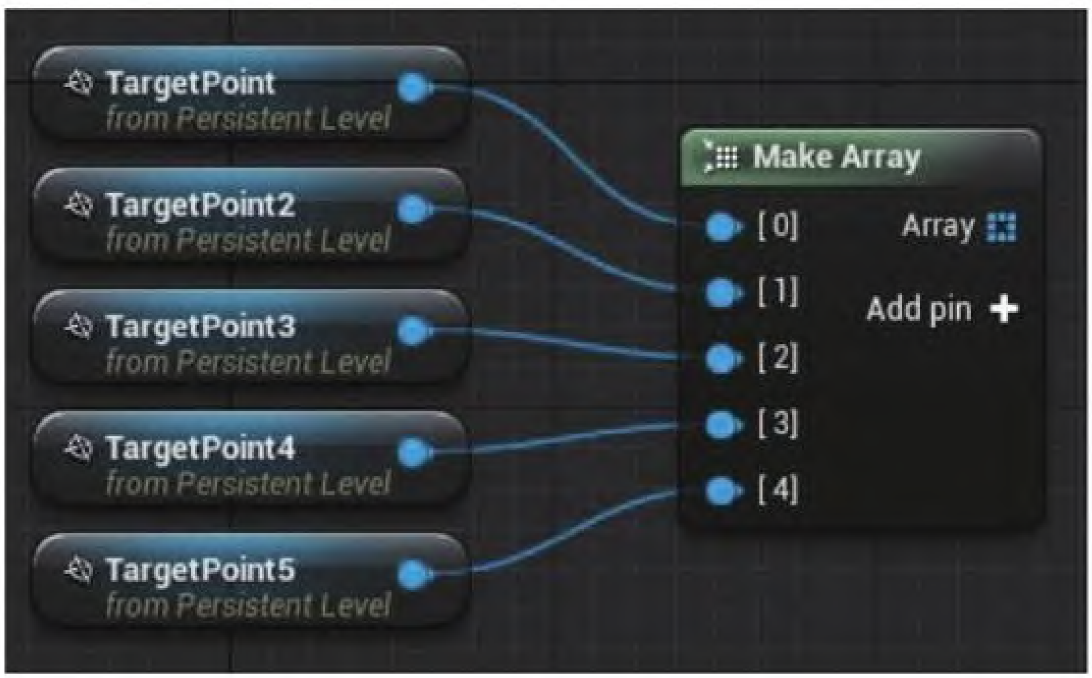



图3-272

**Step 7**　在图表中右击并添加F按键事件，关闭Pressed引脚，添加一个Spawn Actor from Class节点，如图3-273所示。

图3-273

**Step 8**　在MyBlueprint窗口中，创建一个Actor类类型的新变量，如图3-274所示。

**Step 9**　将变量命名为ActorClassArray，单击阵列网格图标，将变量转换为阵列。

**Step 10**　单击Compile按钮，然后在Default Value部分添加3个Actor（使用的是火焰、爆炸和火花效果），如图3-275所示。

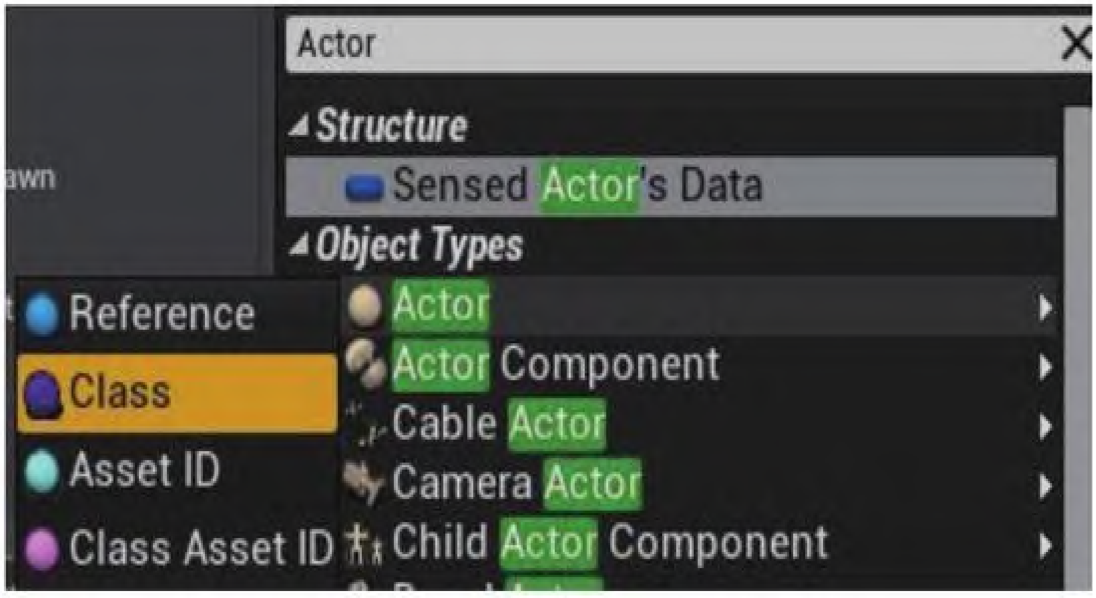

Actor
Structure
Sensed Actor's Data
Object Types
Reference
Class
Asset ID
Class Asset ID
Actor
Actor Component
Cable Actor
Camera Actor
Child Actor Component

图3-274

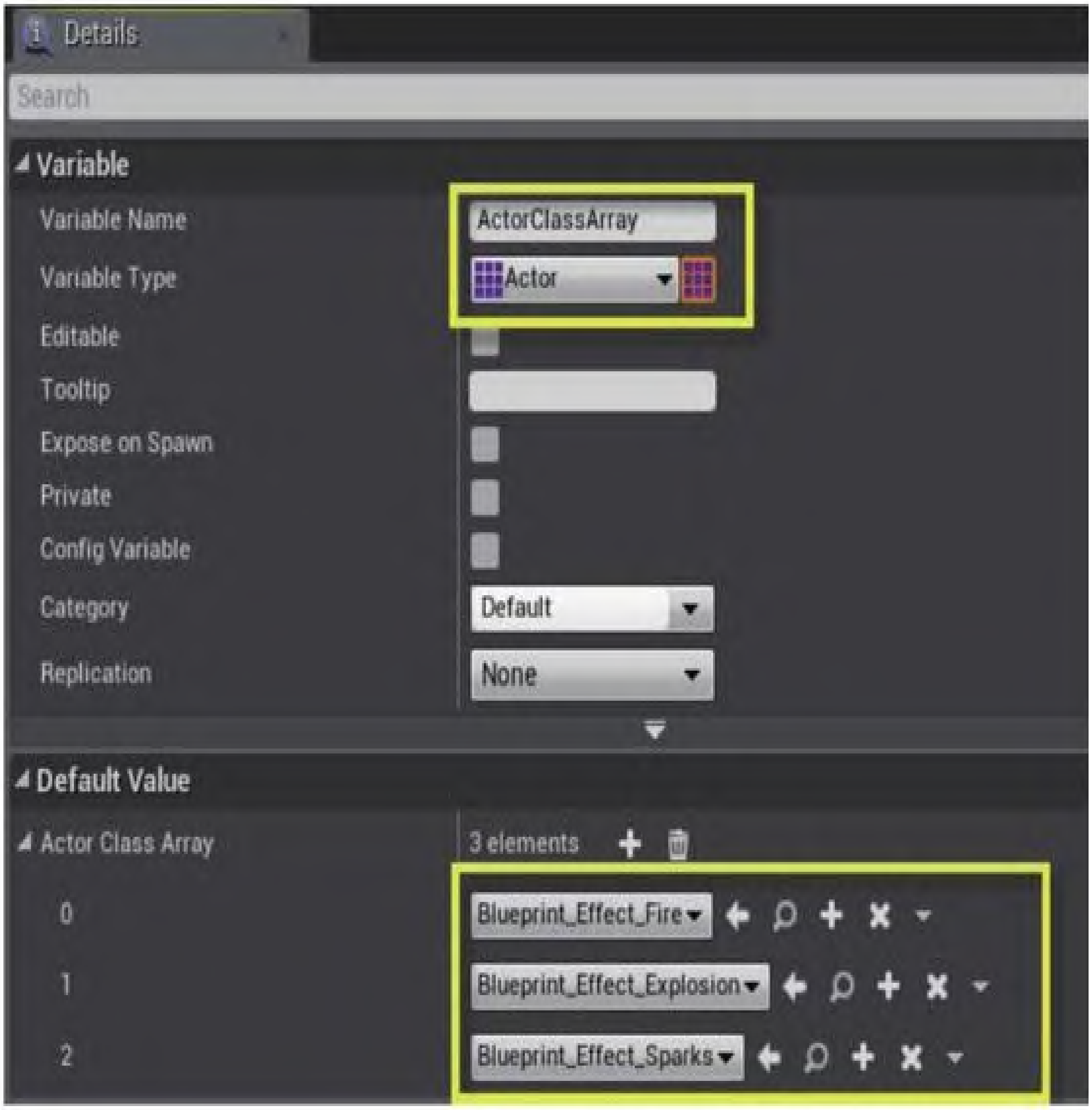



图3-275

**Step 11**　按住Ctrl键拖入ActorClassArray变量，然后拖动它并创建一个Get节点。

**Step 12**　关闭Index引脚，搜索并添加Random Integer in Range节点，如图3-276所示。

因为索引值从0开始，所以将Max值设为2，便可获得每个Actor。

图3-276

**Step 13**　将Max值设为2，然后将Get引脚和Spawn Actor节点上的Class引脚连接起来，如图3-277所示。

图3-277

因为索引值从0开始，所以将Max值设为2，便可获得每个Actor。

Step 14　拖动Make Array节点的输出引脚并添加一个Get节点。

Step 15　关闭Index引脚，再次添加Random Integer in Range节点并将Max值设为4。

Step 16　关闭Get引脚，搜索并添加Get Actor Transform节点，如图3-278所示。

图3-278

Step 17　将Return Value从Spawn Actor节点上的Get Actor Transform引脚连至Spawn Transform引脚。

Step 18　单击Compile和Save按钮，完成后的图表外观如图3-279所示。

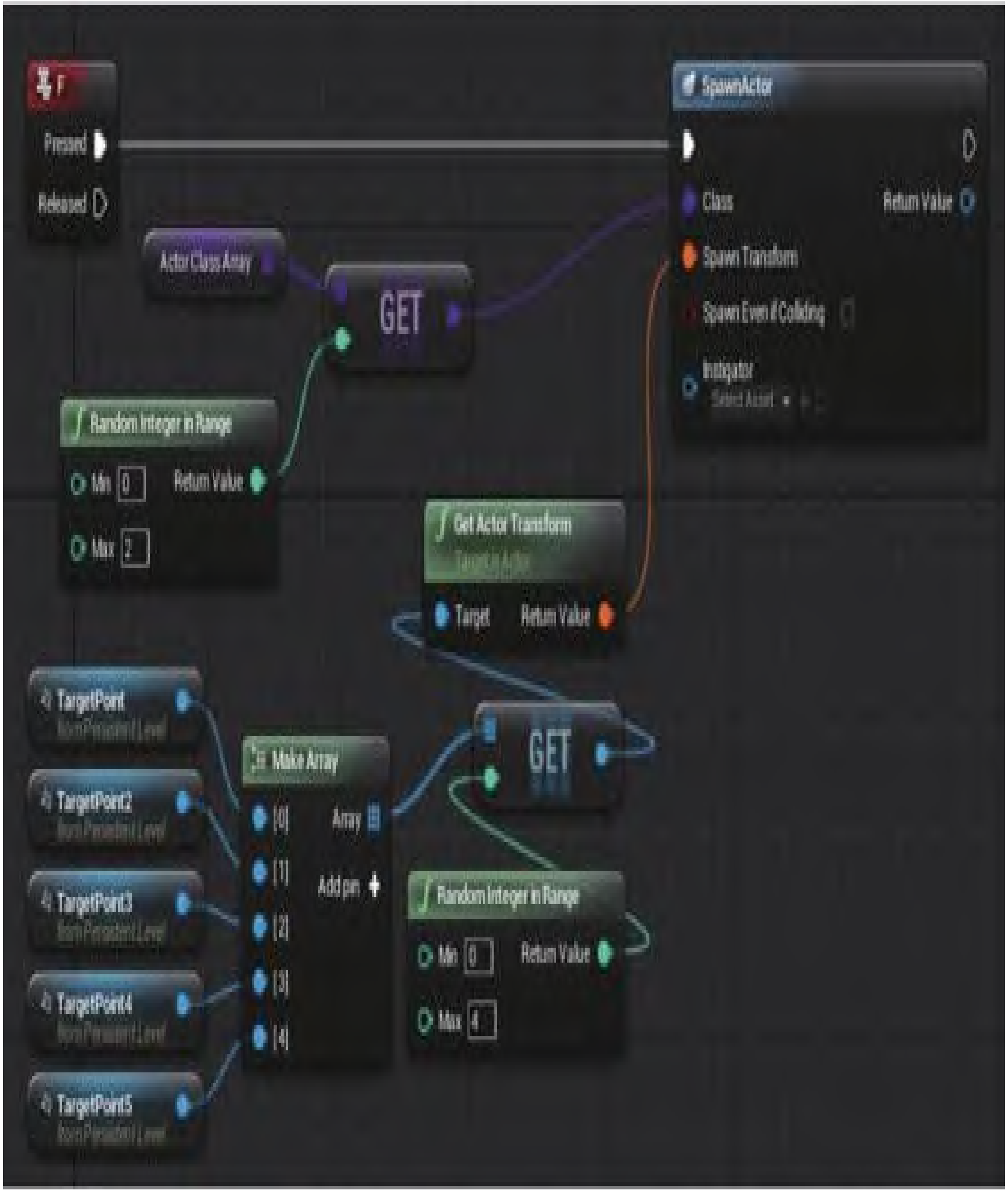



图3-279

**Step 19**　关闭蓝图，然后单击Play按钮在编辑器中开始游戏。

在编辑器中进行游戏时，每次按下F键，一个随机蓝图（从添加至阵列中的蓝图中随机选择）将被选中并在一个位置随机生成（从阵列的目标点中随机选择）。除利用F按键事件生成物品和效果外，还可使用相同逻辑为回复品或弹药随机生成出现地点，甚至掠夺玩家拥有的物品。

## 3.3.13 使用接口

在此节中，我们将设置一个蓝图接口，将其调用后可在多个蓝图中执行不同脚本。

此节使用启用新手内容的Blueprint First Person模板。

**Step 1** 在Content/StarterContent/Props文件夹中，右击SM_GlassWindow按钮并选择Create Destructible Mesh，如图3-280所示。

我们需要将静态网格体转换为可摧毁网格体（Destructible Mesh），使其在被击中后能够破碎。

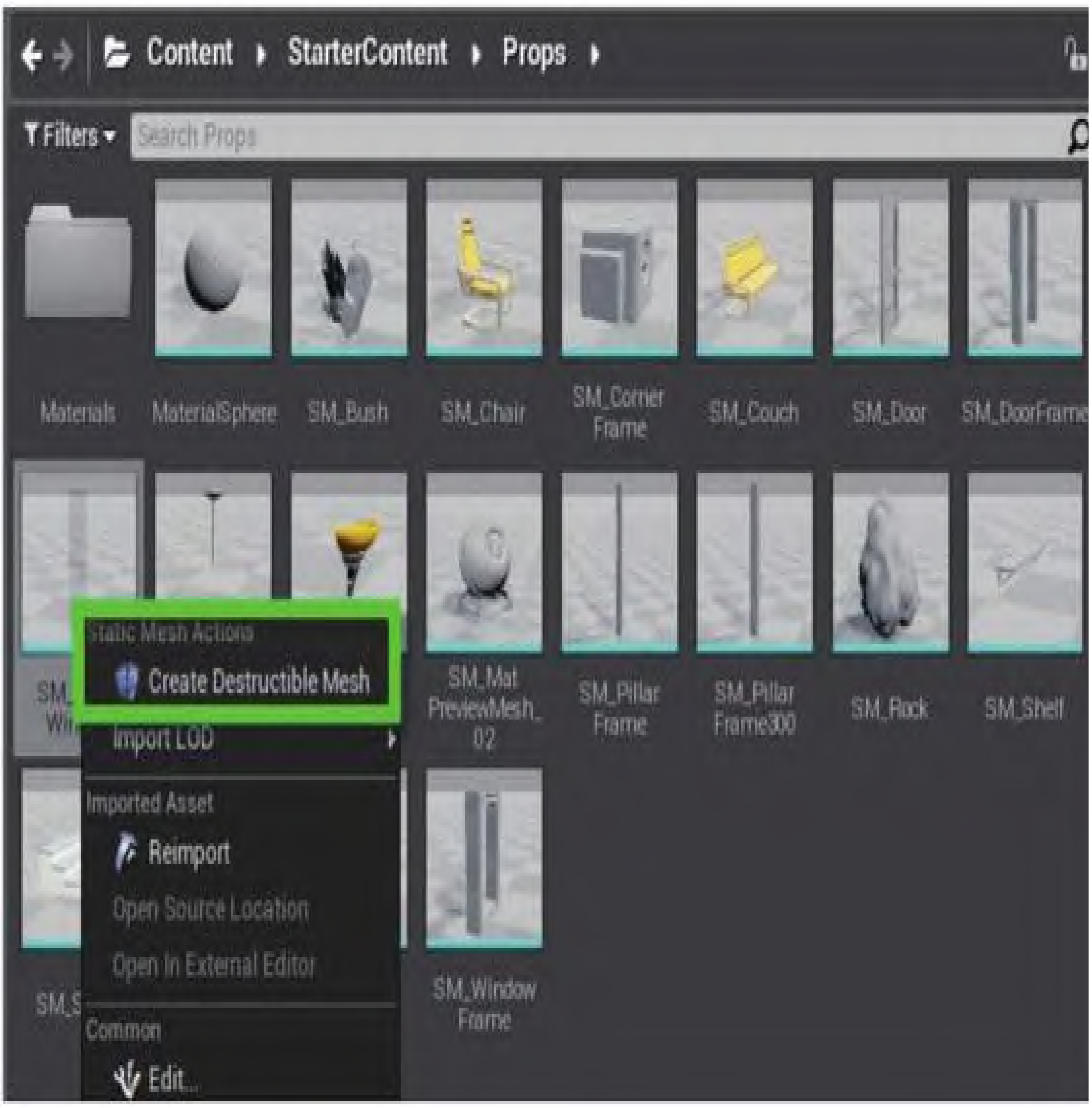



图3-280

**Step 2** 打开可摧毁网格体，单击Fracture Mesh和Save，然后关闭可摧毁网格体编辑器窗口，如图3-281所示。

图3-281

**Step 3** 右击SM_GlassWindow_DM资源，然后在Asset Actions下选择Create Blueprint Using This，如图3-282所示。

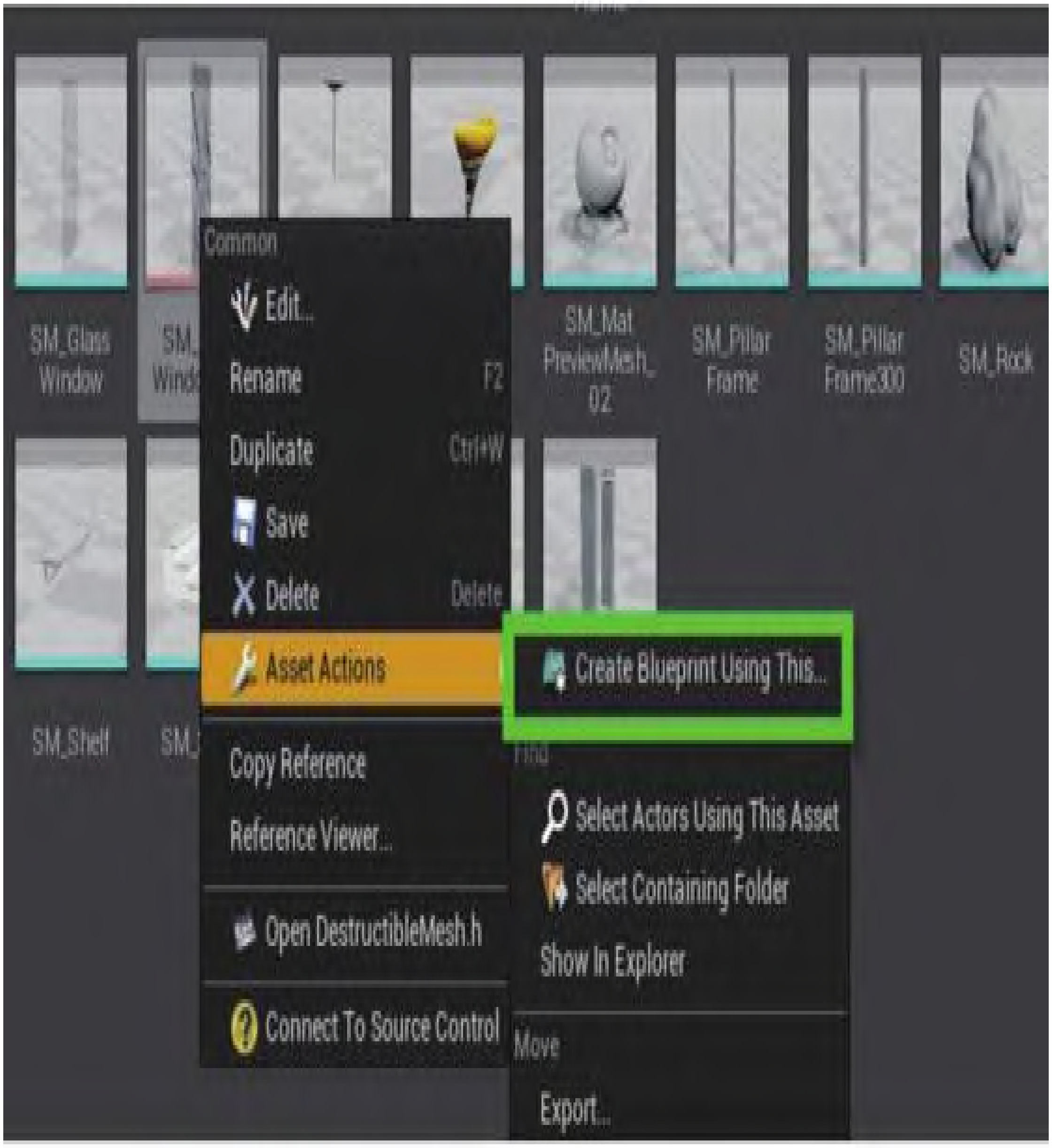



图3-282

将弹出提示输入蓝图的命名和保存路径（此处使用SM_GlassWindow_DM）的信息。

**Step 4**　在内容浏览器中右击，然后在Blueprints下选择Blueprint Interface，如图3-283所示。

将再次弹出提示输入蓝图接口的命名和保存路径（此处使用DM_Interface）的信息。

Step 5　在蓝图接口中，将现有函数命名为IsDestructible？，然后单击New按钮添加下图中显示的输入（Inputs），如图3-284所示。

图3-283

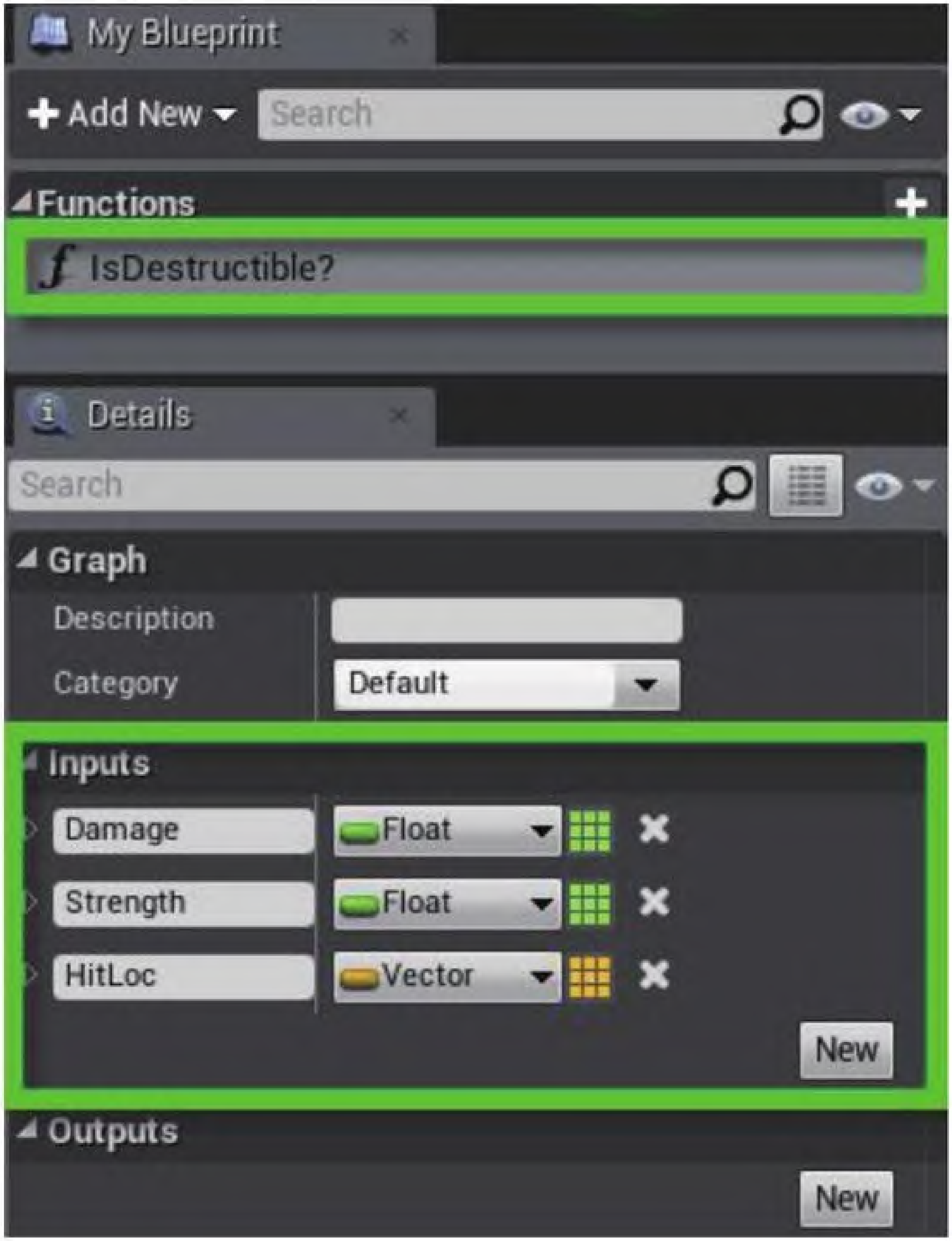



图3-284

上图中我们创建了三个Inputs，用于在应用此蓝图接口的蓝图之间传递变量。我们创建了两个浮点变量：一个名为Damage，用于传递命中时对物体形成的破坏力量；另一个名为Strength，用于传递应用的冲击力量。最后添加一个名为HitLoc的矢量，用于保存物体被命中的位置，以便传递信息。

**Step 6** 编译并保存，然后关闭蓝图接口。

**Step 7** 在内容浏览器中打开SM_GlassWindow_DM蓝图，然后单击工具栏中的Class Settings选项，如图3-285所示。

图3-285

**Step 8** 在Details面板中，单击Interfaces下的Add按钮并添加DM_Interface，如图3-286所示。

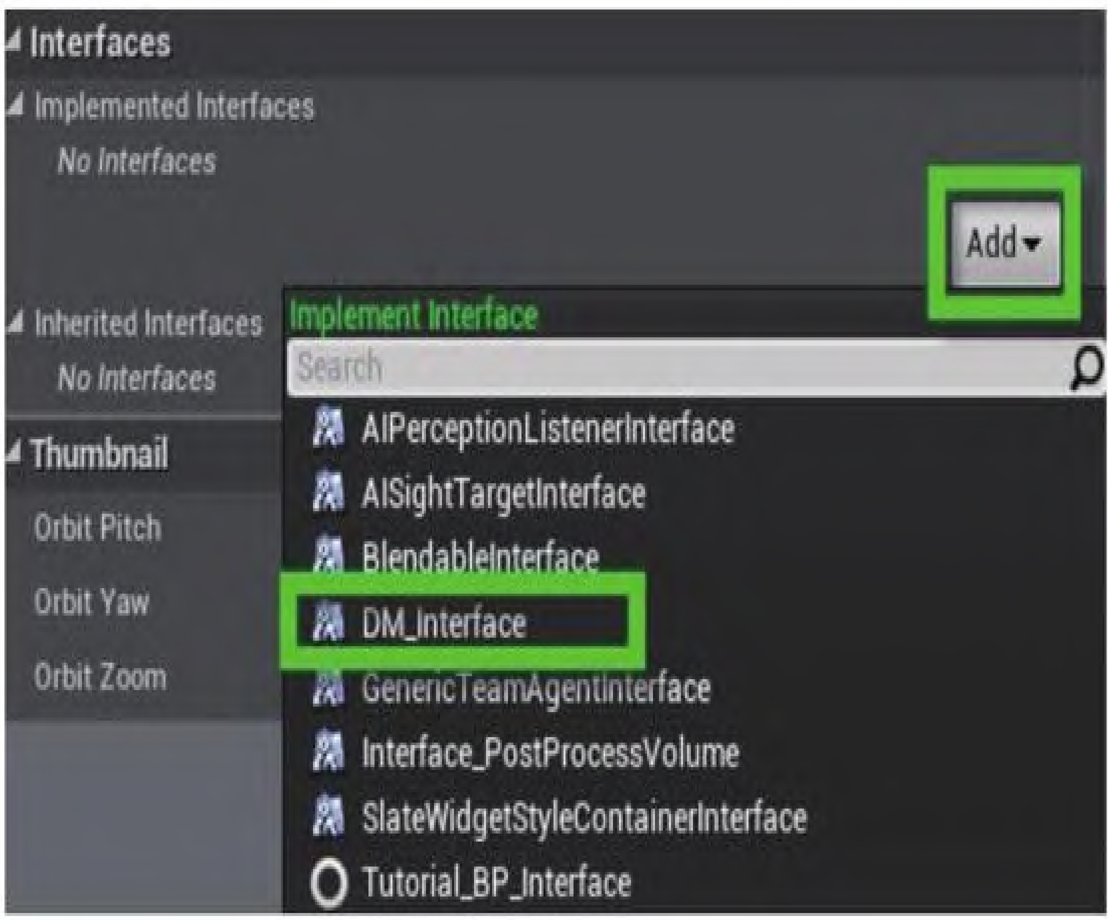



图3-286

此操作将在此蓝图中应用蓝图接口，以便访问其中的函数。

**Step 9** 在图表中右击，搜索“？”并选择Event Is Destructible?事件，如图3-287所示。

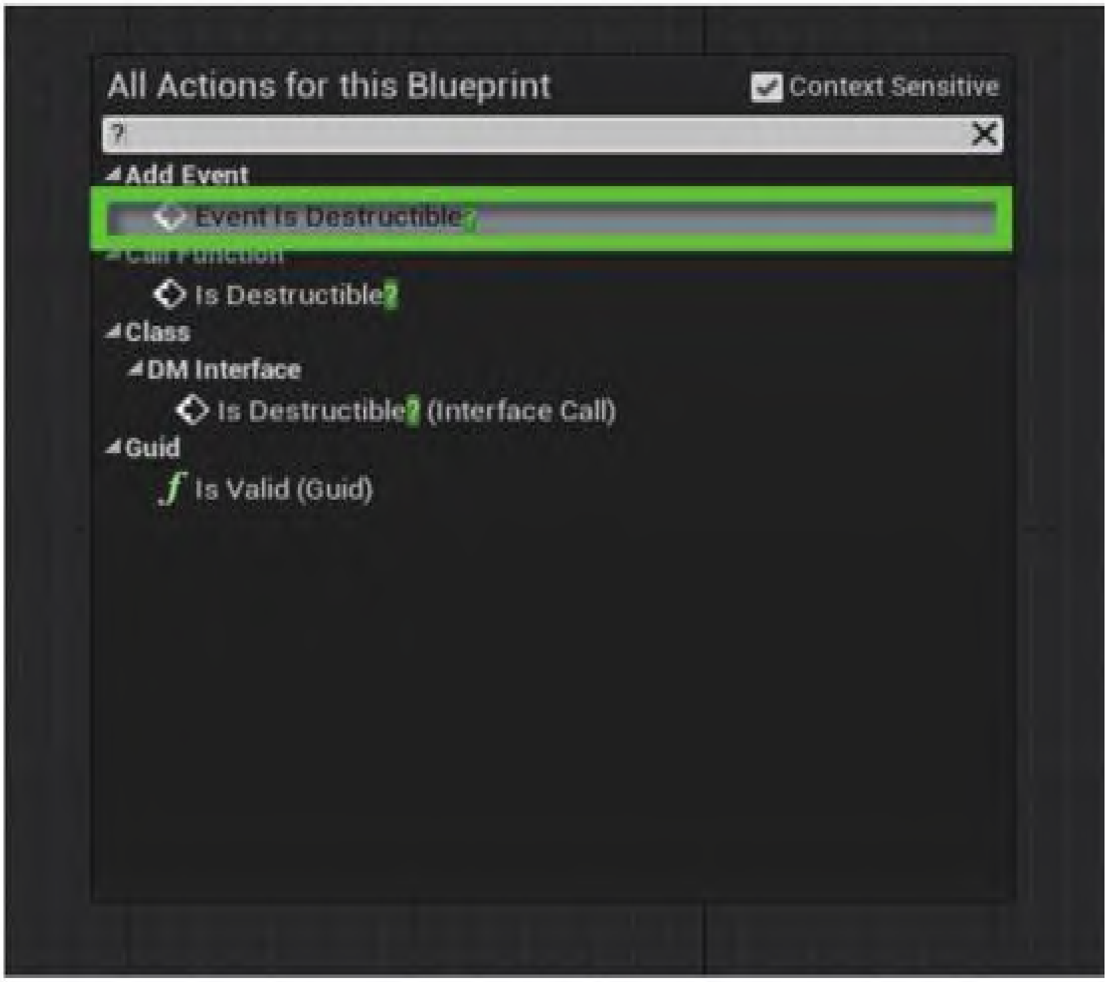



图3-287

**Step 10** 长按Ctrl键从MyBlueprint窗口拖入Destructible组件并创建以下脚本，如图3-288所示。

图3-288

在Destructible组件引出连线后，我们使用Apply Damage并连接所需的损坏、力量、位置和方向要求，对玻璃窗应用破坏。实际的损坏、力量和命中位置信息将通过蓝图接口从另一个蓝图（稍后进行设置）传递到此蓝图。

**Step 11**　编译并保存，然后关闭蓝图。

**Step 12**　打开内容浏览器中Content/StarterContent/Blueprints下的Blueprint_Ceiling Light蓝图，如图3-289所示。

图3-289

对此灯进行射击时，我们将使用蓝图接口执行不同的操作。

**Step 13** 通过Class Settings添加DM_Interface，然后拖入PointLight1组件并创建以下脚本，如图3-290所示。

图3-290

Event Is Destructible? 被调用时，我们首先将灯光设为绿色。第二次调用时，我们将把强度设为0（关闭灯光）。注意：我们并未使用接口中包含的损坏、力量或命中位置信息，因其与此蓝图无关。使用蓝图接口可形成一个整体函数，在应用接口和函数的所有蓝图中进行调用，然后确定每个蓝图上执行的脚本。

**Step 14** 编译并保存，然后关闭蓝图。

**Step 15** 打开内容浏览器中Content/FirstPersonBP/Blueprints下的FirstPersonProjectile蓝图，如图3-291所示。

此蓝图包含玩家武器开火时游戏中生成的投射物的脚本。

**Step 16** 通过Class Settings添加DM_Interface，然后在图表中右击并添加IsDestructible?（Message）节点，如图3-292所示。

Content
FirstPersonBP
Blueprints
Filters
Search Blueprints
FirstPerson Character
FirstPerson GameMode
FirstPerson HUD
FirstPerson Projectile

图3-291

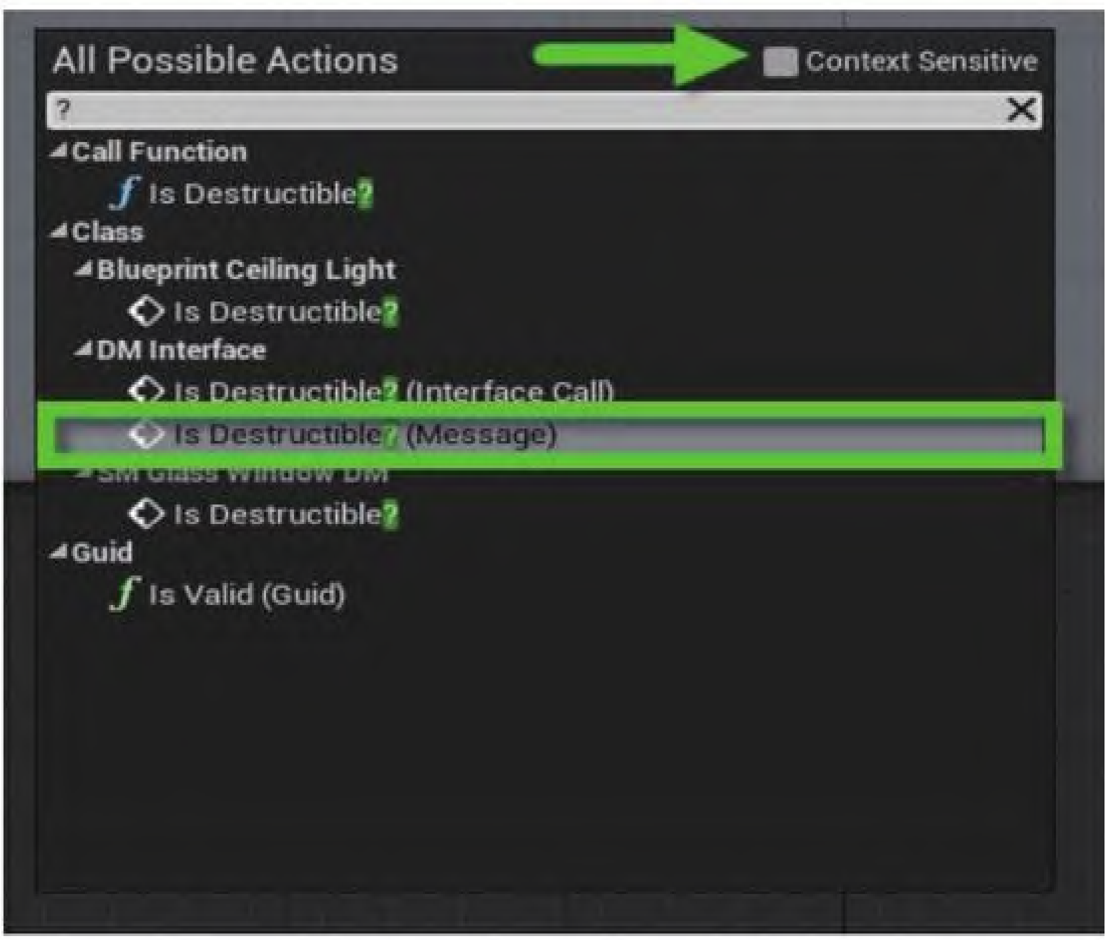

All Possible Actions
Context Sensitive
?
Call Function
Is Destructible?
Class
Blueprint Ceiling Light
Is Destructible?
DM Interface
Is Destructible? (Interface Call)
Is Destructible? (Message)
Is Destructible?
Guid
Is Valid (Guid)

图3-292

此操作将执行函数IsDestructible？，应用接口和函数的所有蓝图也将执行。

如未找到此节点，可取消勾选快捷菜单右上角的Context Sensitive选项。

**Step 17** 在Branch False引脚引出连线后连接IsDestructible？节点，并连接Other和HitLocation引脚，如图3-293所示。

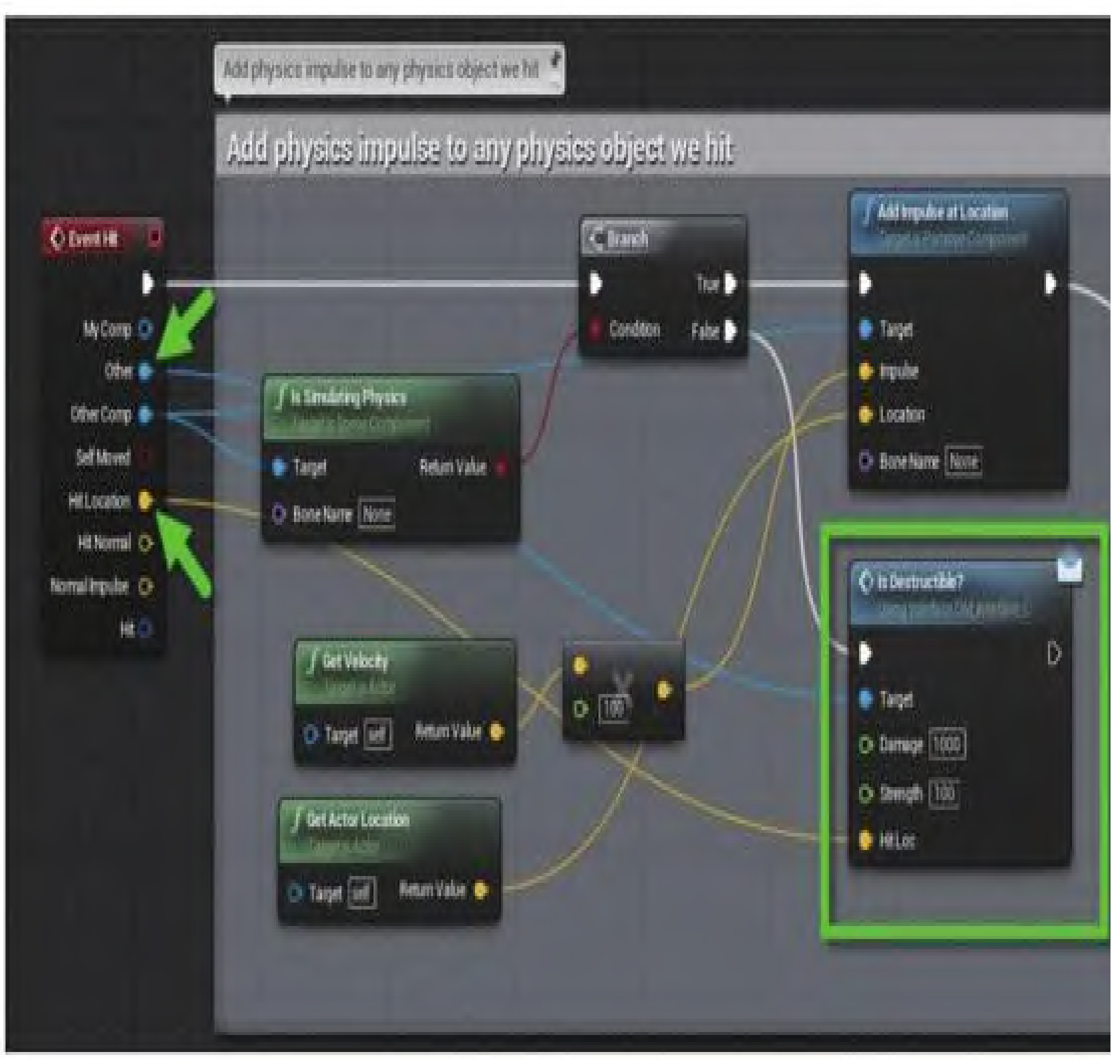



图3-293

在上图中，我们添加了现有的脚本，检查命中的物体是否进行了物理模拟。如未进行物理模拟，则在蓝图接口中调用IsDestructible？函数，并为目标应用1000损坏和100力量（从EventHit节点的Other引脚收集，在此引脚处还可获得命中位置并指定到HitLoc变量进行传递）。

**Step 18**　编译并保存，然后关闭蓝图。

**Step 19**　将SM_GlassWindow_DM和Blueprint_CeilingLight蓝图拖入关卡，然后单击Play按钮在编辑器中开始游戏。

射击玻璃将使其碎裂，如图3-294所示。

图3-294

对灯射击一次，将使灯光变为步骤13中指定的颜色，如图3-295所示。

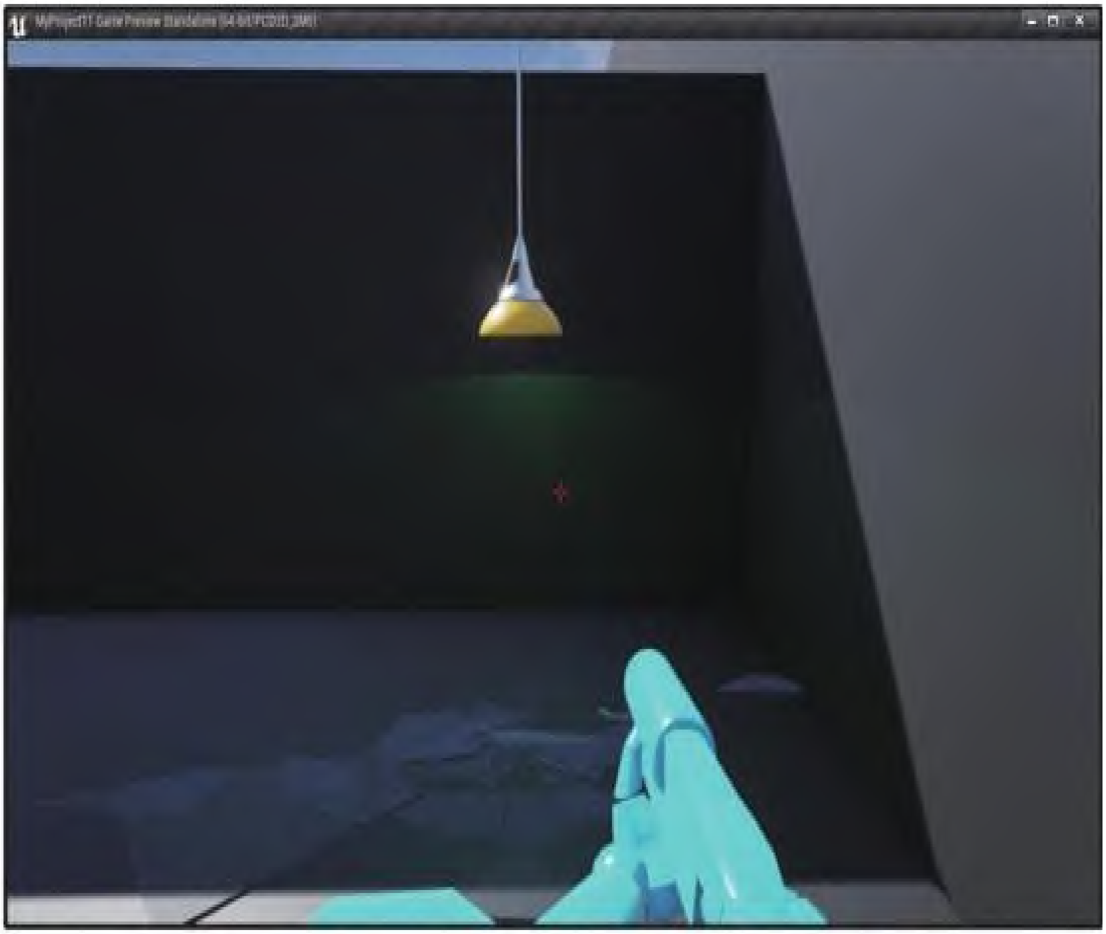


图3-295

再次对灯进行射击，将使灯光关闭（已在步骤13中设置），如图3-296所示。

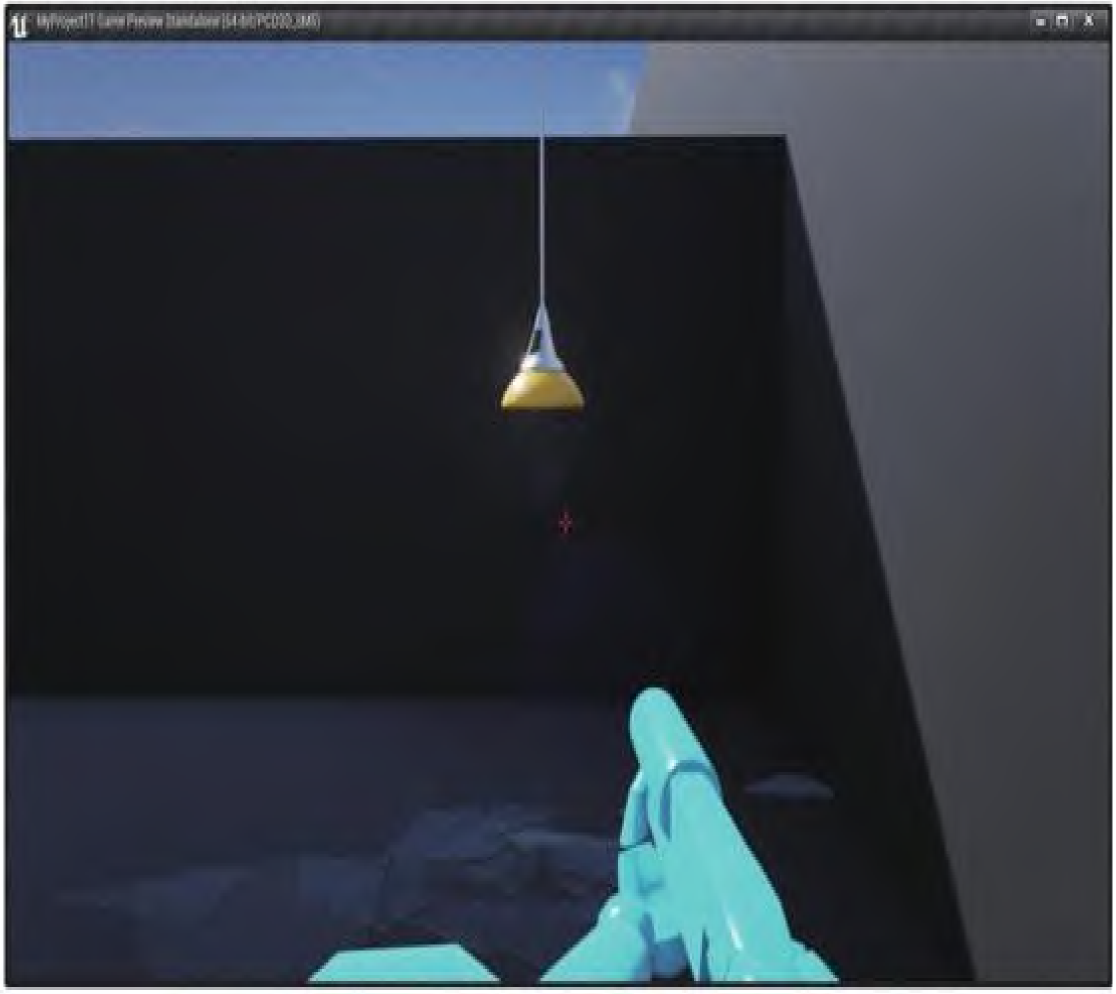


图3-296

此例使用蓝图接口在蓝图之间传递变量，并调用在应用接口的蓝图中执行不同脚本的函数。

## 3.3.14 使用宏库

宏库是一个容器，存放有大量宏或自含式图表（可作为节点放置到其他蓝图中）。它们保存常用节点顺序（包括执行和数据传输的输入和输出），因此可有效节约时间。

创建宏库：

此例中我们创建由两个宏组成的宏库，一个用于添加“体力值”变量，两一个用于调整指定Actor的大小。

此例中我们使用的是启用新手内容的Blueprint Third Person项目。

**Step 1**　在内容浏览器的空白处右击，然后在Blueprints的快捷菜单中选择Blueprint Macro Library，如图3-297所示。

图3-297

**Step 2**　在出现的Pick Parent Class窗口中选择父类（此例中选择Actor），如图3-298所示。

**Step 3**　为宏库命名，然后双击打开，如图3-299所示。

将显示蓝图宏界面，如图3-300所示。

Step 4　在右上角的MyBlueprint窗口中，将默认宏上按下F2将其重命名为ScaleUp。

Step 5　在ScaleUp宏的Details面板中添加Inputs和Outputs，如图3-301所示。

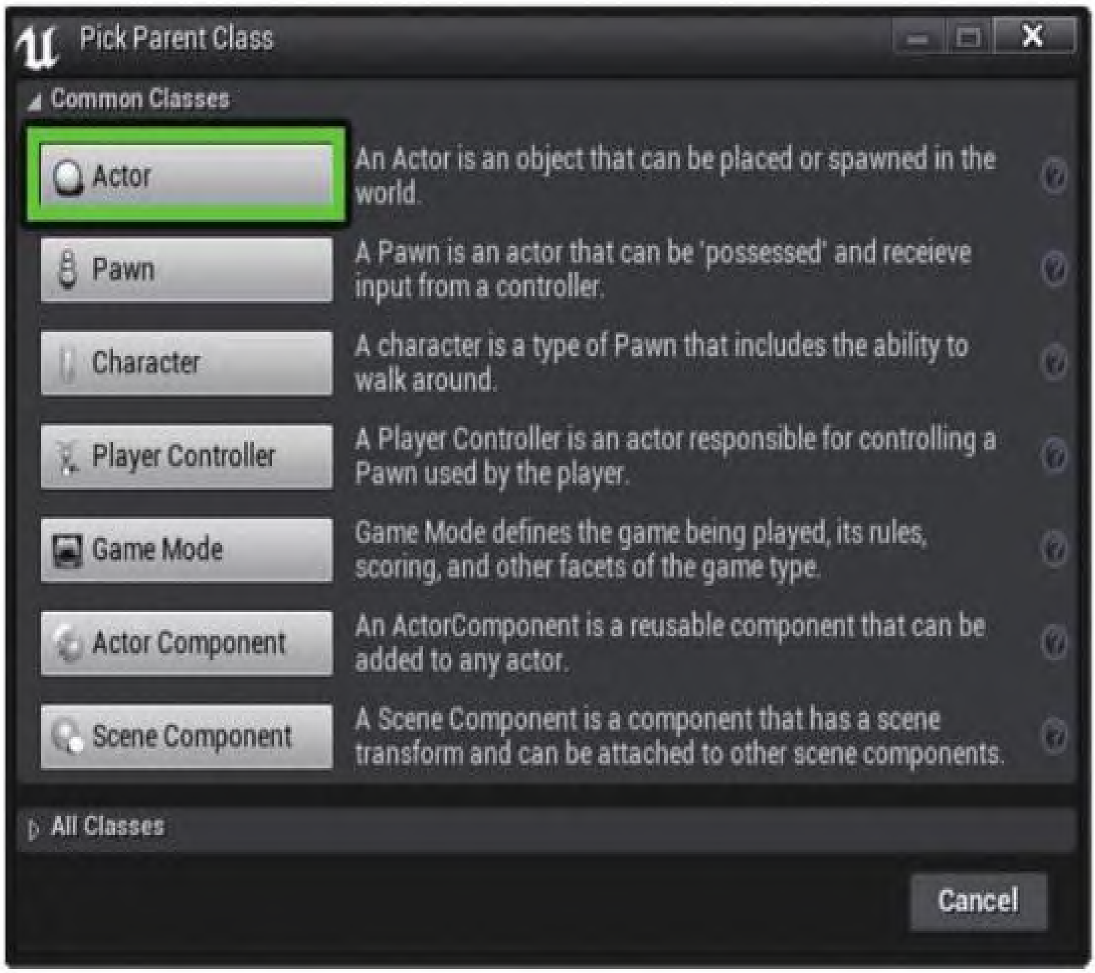



图3-298

Filters
Search Blueprints
MyMacroLibrary
ThirdPerson
Character
ThirdPerson
GameMode

图3-299

MyMacroLibrary
File
Edit
Asset
View
Debug
Window
Help
Parent class: Actor
Search For Help
Compile
Save
Find in CB
Search
Class Settings
Class Defaults
Play
My Blueprint
Add New
Search
Macros
NewMacro_0
New Macro 0
MyMacroLibrary > NewMacro_0
Drag Off Pins to Create/Connect New Nodes.
Inputs
Outputs
Details
Search
Graph
Description
Category
Default
Instance Color
Inputs
New
Outputs
New
MACRO

图3-300

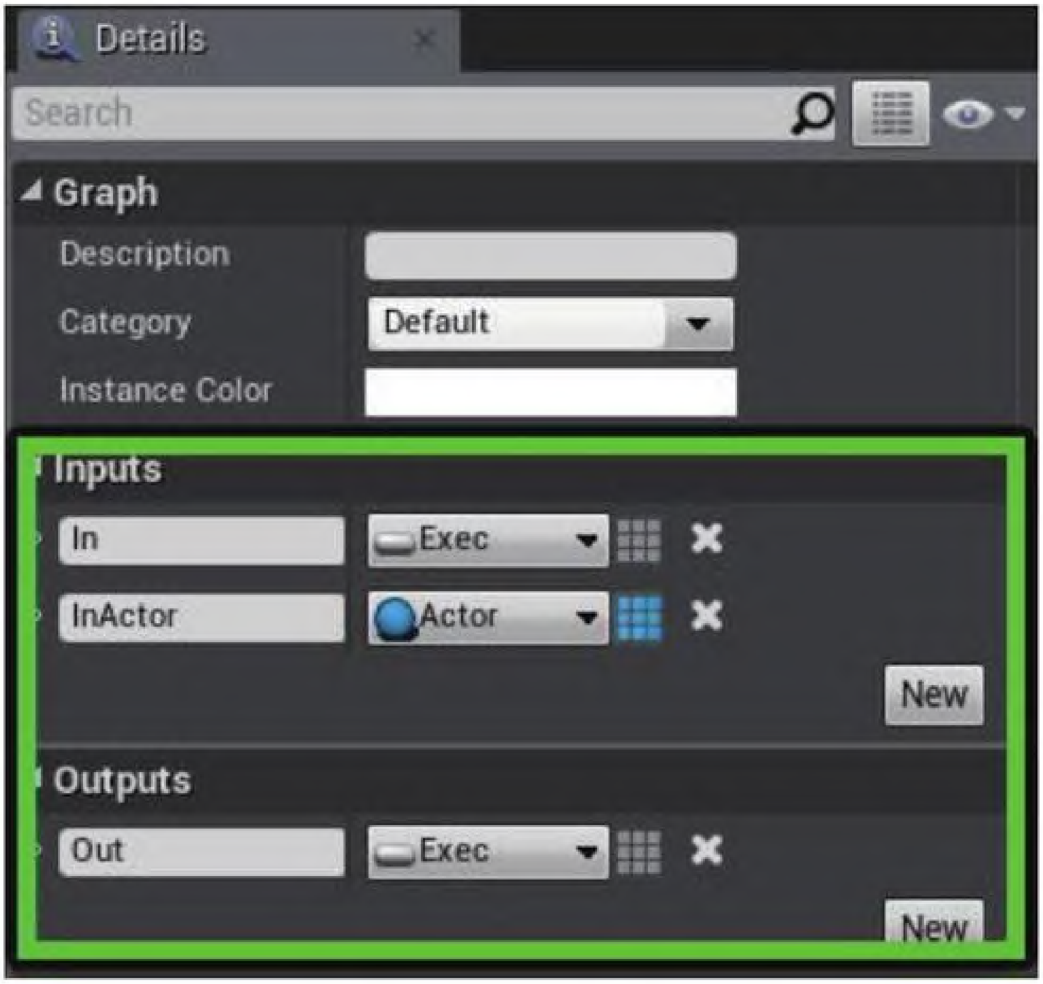



图3-301

上图中创建了两个输入：一个名为In，设为Exec类型；另一个名为InActor，设为Actor类型。添加的输出名为Out，设为Exec类型。我们将使用它们来决定需要通过此宏进行影响的Actor。

**Step 6**　在ScaleUp宏的图表中，按下图重建节点网络，如图3-302所示。

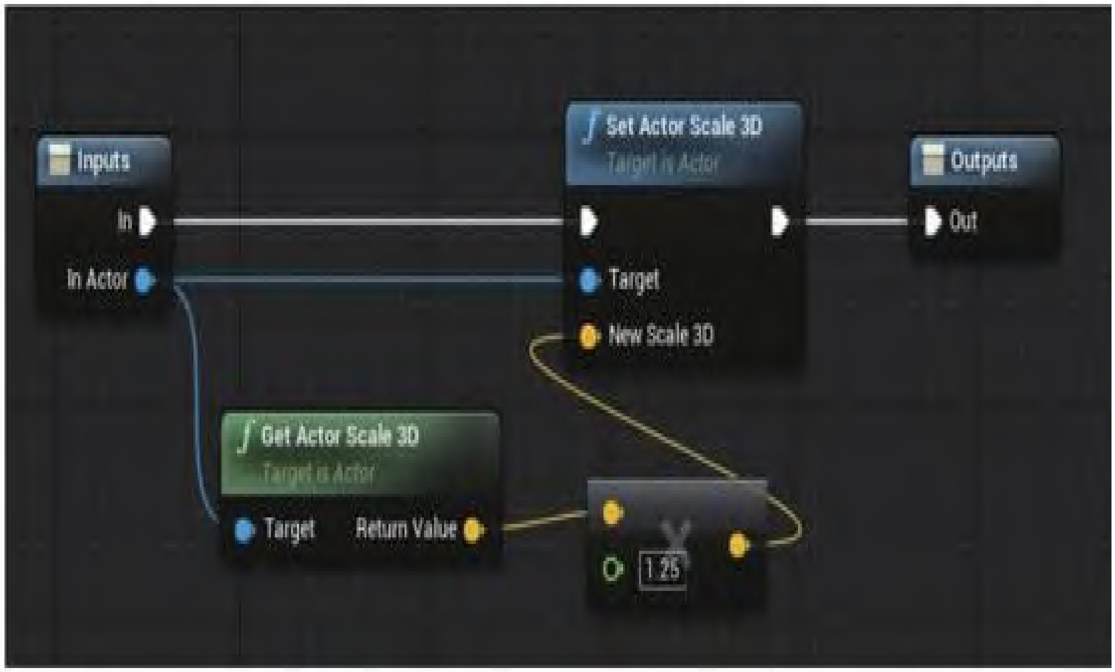



图3-302

在上图中，此宏被调用时，它将获取Actor作为InActor提供的内容；并获取其当前大小，在将其设为目标（即为指定为InActor的Actor）的新大小3D之前乘以1.25。我们现在即可结合任意Actor使用此宏，并使用此宏调整Actor的大小。

**Step 7**　从MyBlueprint窗口添加另一个名为AddHealth的宏。

**Step 8**　在AddHealth宏的Details面板中添加Inputs和Outputs，如图3-303所示。

上图中我们使用一个名为GetHealth的浮点值，通过SetHealth输出节点进行传递之前将把此值添加到该节点。

**Step 9**　在AddHealth宏的图表中，按下图重建节点网络，如图3-304所示。

上图中，在输出结果值到SetHealth之前，为作为GetHealth值的浮点值增加50。

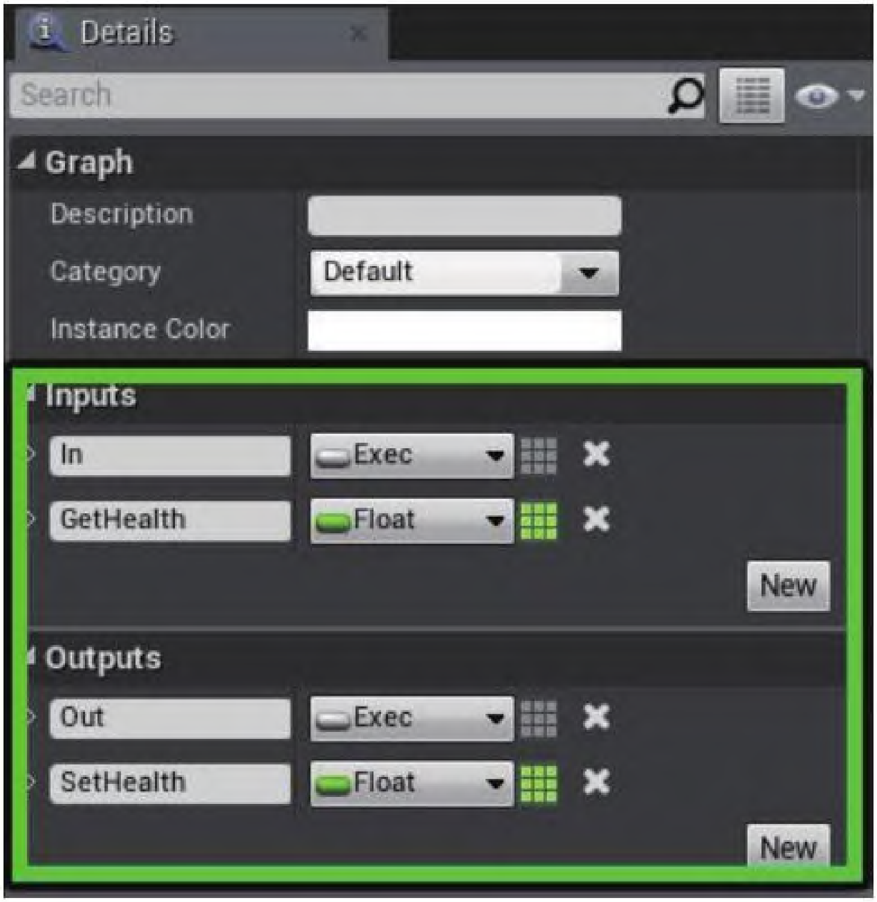



图3-303

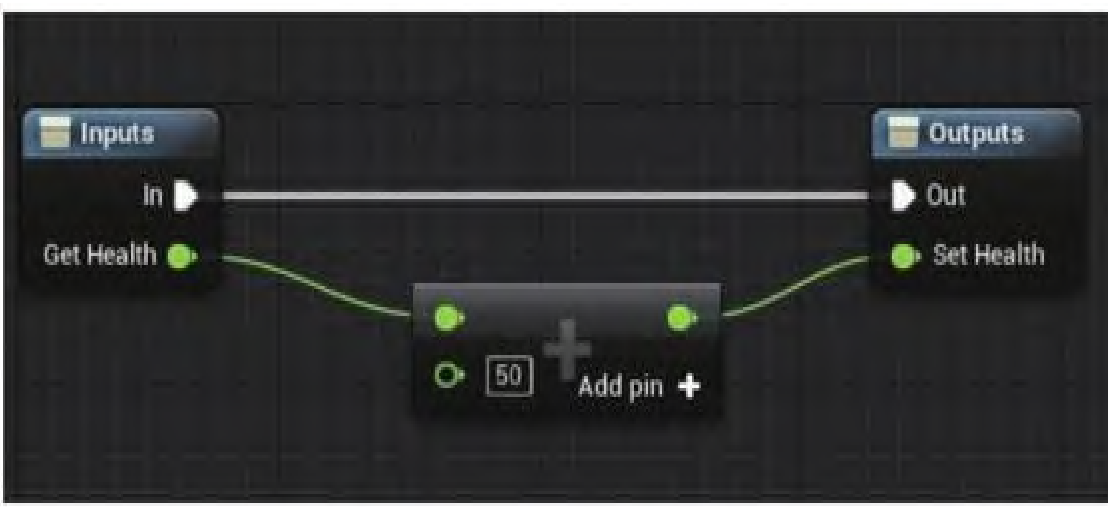



图3-304

上图中，在输出结果值到SetHealth之前，为作为GetHealth值的浮点值增加50。

**Step 10**　保存并关闭蓝图宏库。

**Step 11**　在Content/ThirdPersonBP/Blueprints文件夹中，打开ThirdPersonCharacter蓝图。

**Step 12**　在MyBlueprint窗口中添加一个代表Health的浮点变量（将默认值设为0），如图3-305所示。

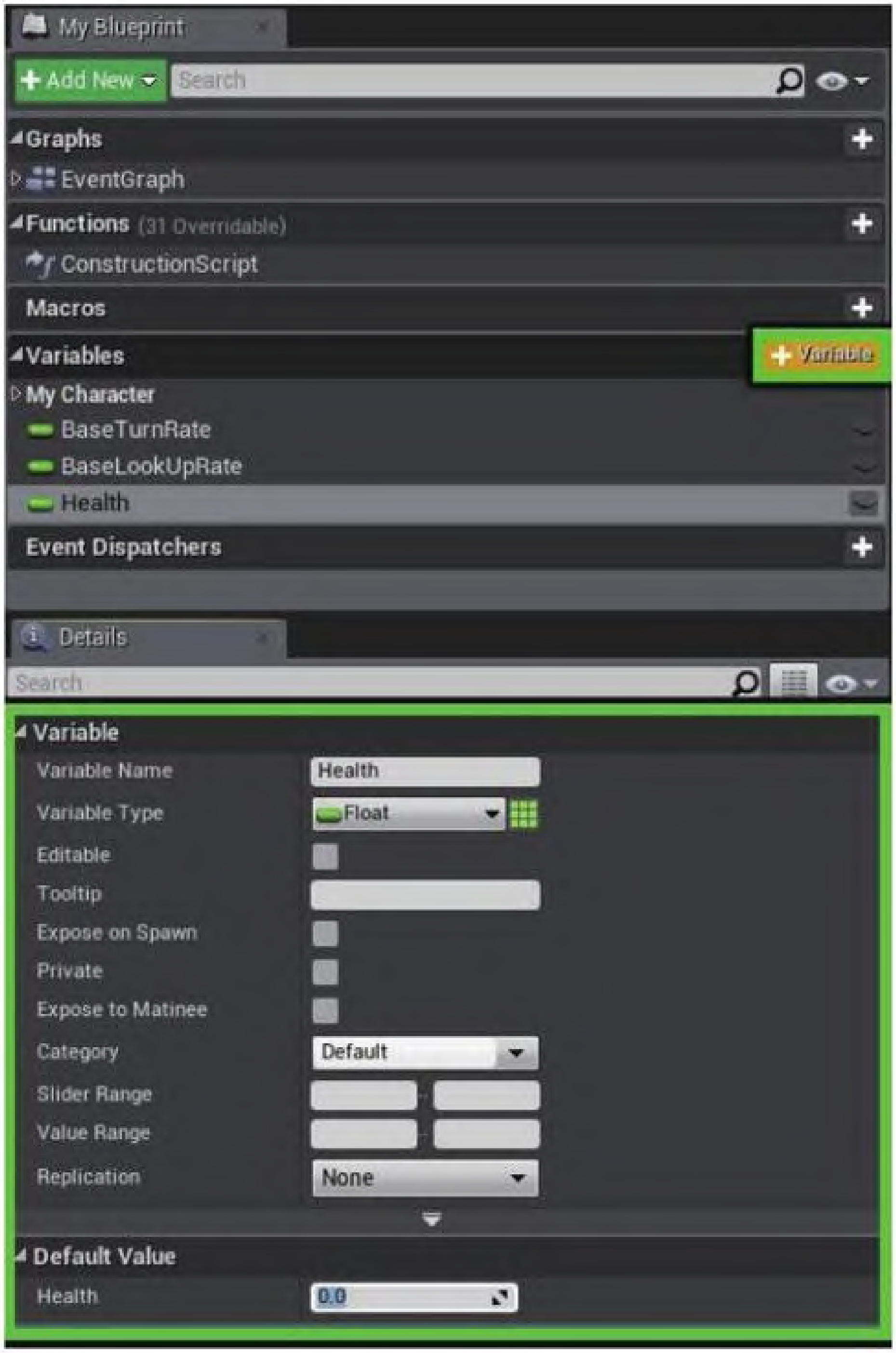



图3-305

**Step 13**　在事件图表中，重建节点网络，如图3-306所示。

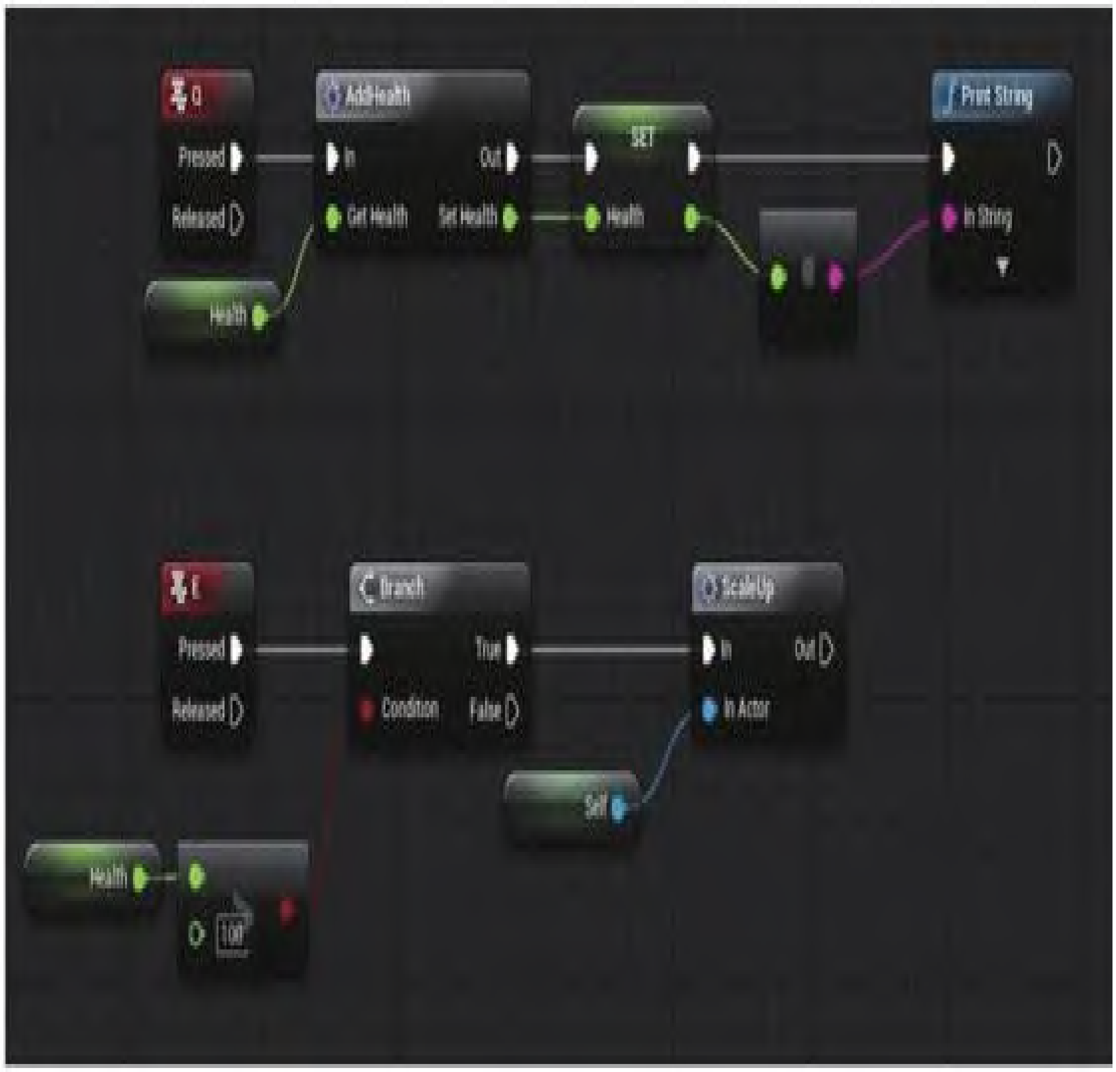



图3-306

上图中，按下Q键后将调用AddHealth宏，它将抓取Health变量（执行增加的宏脚本）并通过输出引脚SetHealth进行更新，然后再显示到屏幕上。我们先使用E按键事件检查Health是否大于100，然后允许ScaleUp宏在ActorSelf（即为ThirdPersonCharacter）上调用。

**Step 14** 编译并单击Play在编辑器中开始游戏。

按下E键尝试调用ScaleUp宏时，首先检查默认设为0的Health值。之后按下Q键调用AddHealth宏，在每次按键时为Health变量增加50。数次按键后，再次尝试调用ScaleUp宏便会成功，因为增加后Health已经大于100。

这些宏可从提供Health输入的其他蓝图和提供的目标Actor进行调用。

3.4

# 蓝图最佳方法

通过蓝图可执行许多操作。从制作小游戏或程序化内容工具，到设计新功能原型，再到调试和改进程序员制作的内容，均可通过蓝图可视化脚本系统完成。

然而，如果在蓝图中完成部分特定操作，将对性能产生较大影响。如果蓝图在每个tick都执行大量操作和复杂的数学计算，则需要考虑使用本地C++代码。蓝图最适合创建事件驱动的功能，如处理伤害接收、控制，或其他非每帧调用的操作。

即使你的技巧非常适合于蓝图，在实际的蓝图设置过程中仍然会面临一些决定，这些决定将使过程更加顺畅。此指南将讲述部分常见决定，以及对蓝图使用者的提示和技巧。

## 3.4.1　关卡蓝图vs蓝图类

使用过UE3 Kismet的使用者不会对关卡蓝图感到陌生，因为可在关卡中直接选择物体并进行操作。它们适合创建一次性原型和熟悉蓝图系统，但只限定于它们使用中的关卡。这意味着关卡蓝图非常利于对关卡限定功能，或对其中的Actor进行设置。在一些范例中，触碰到特定开关时将启动过场，或在消灭所有敌人后打开一扇特定的门。

总之，蓝图类是在项目中实现可重复行为的最佳方式。创建蓝图类后，可将其添加到任意关卡，还可随意添加任意数量的副本到关卡中，无须四处复制脚本。

如你从关卡蓝图开始，然后决定将行为移入蓝图类，此过程将会非常简单。当你从关卡蓝图复制在Actor上进行操作的函数（如在一盏灯上调用Set

Brightness），并基于灯光Actor的关闭将它们复制到蓝图类中时，函数将更新到适当的范围中。

### 3.4.2　类变量vs本地变量

在蓝图中进行工作时，可使用我的蓝图标签添加变量。在函数中可看到本地变量My Blueprint标签下的额外部分，如图3-307所示。

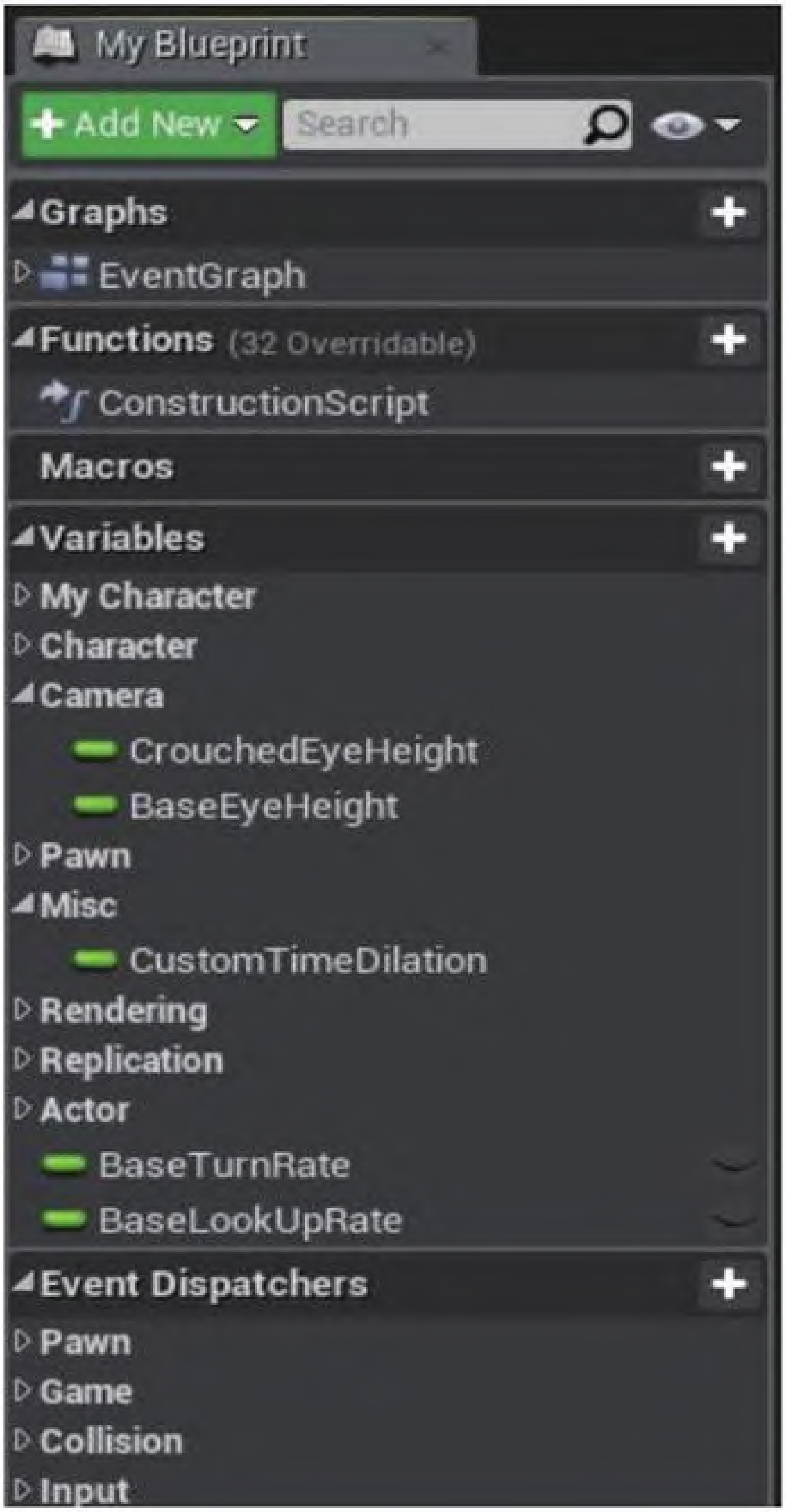

My Blueprint
Add New
Search
Graphs
EventGraph
Functions (32 Overridable)
ConstructionScript
Macros
Variables
My Character
Character
Camera
CrouchedEyeHeight
BaseEyeHeight
Pawn
Misc
CustomTimeDilation
Rendering
Replication
Actor
BaseTurnRate
BaseLookUpRate
Event Dispatchers
Pawn
Game
Collision
Input

图3-307

本地变量拥有适用范围，意味着它们只会出现在定义的范围中。因此，函数中的本地变量仅限该函数可见，其他函数不可见，事件图表也不可见。这有助于减少仅在函数上下文中相关的内容出现的混乱。在函数中使用时，可将本地变量视为“便笺纸”。它们可帮助函数执行操作，函数完成之后即被废弃。

需要从蓝图中多处进行访问的内容即为类变量。因此，诸如对网格体或其他组件的引用，或需要公开以便从其他蓝图访问的变量等均为类变量。使用之后你便会对其留下深刻印象。

### 3.4.3　函数vs宏

函数和宏两者在外部完成相同目标：它们发送输入到节点后，事件发生，然后进行输出。它们以两种不同方式进行，但也存在一些共性：

•两者均有用于编辑节点功能的中心区域（宏拥有宏图表，函数拥有函数图表）。中央区域发生变化后，调用函数或宏的每个节点均会更新。

•两者均可重复使用。

•两者均可使用本地变量。宏使用没有特定的命名，只保存数值的“匿名”本地变量。

•两者均适用于封装重复使用和清晰性功能。

在更高级的使用情况中将了解到一些主要不同点。

•放置节点调用函数时，函数将被实际调用。意味着可以函数为目标（如“在另一个对象上调用函数”），而函数则形成蓝图之间的通信。

•宏接受来自宏图表的节点，并用所有这些节点的副本实际替换宏节点。根本而言，蓝图被编译时，宏将复制所有图表节点，并将它们粘贴到宏节点所在之处。

因它们在后台工作的方式，意味着可进行的操作中存在一些不同：

•宏可对其范围设定的类（新建宏蓝图时选择的类，或定义本地宏的类）使用任意节点。这意味着放入宏的节点拥有更强的通用性。

•函数和宏之间最大的差异之一是隐藏节点可放置在宏中，但不可放置在函数中。

•可覆写子蓝图中的函数功能。例如一个汽车蓝图拥有一个“PlayerInteractedWithMe”函数。在此函数中可按响汽车喇叭。现在假设汽车蓝图拥有两个子项，一个是警车，一个是消防车。可在警车中覆写此函数，使其播放警笛声并开启灯光。也可在消防车中覆写此函数，使其喷水。宏无法实现此类覆写功能。

•因为宏只在编译时才复制粘贴到图表中，一个宏上可存在多条进出的执行引线。无法对函数执行此操作。

总之，如果需要在各处重复使用一些功能，最好使用宏。但如果需要调整蓝图子项中的行为，或需要从另一个蓝图直接进行访问，则最好使用函数！

## 3.4.4　蓝图通信类型

两个蓝图之间存在几种不同的通信方法。最常见的使用情况是直接蓝图通信，有时需要实现的功能意味着你将使用事件分配器或蓝图接口替代。如需了解每种通信类型的概览、使用情况范例，以及上手教程，请查阅蓝图通信用法。如需创建使用三种类型的项目，请查阅蓝图通信项目指南。

## 3.4.5　常规提示

### 3.4.5.1　在蓝图编辑器中工作

在蓝图编辑器中工作时，可通过几种技巧寻找到需要使用的节点，以及已创建的节点和注释。

•在现有蓝图中寻找内容时（无论是变量、函数或注释），可使用蓝图编辑器中的Search按钮。如取消勾选Find in Current Blueprint Only框，可对所有蓝图进行搜索（甚至未加载的蓝图），便于针对应用内容之处进行追踪，如图3-308所示。

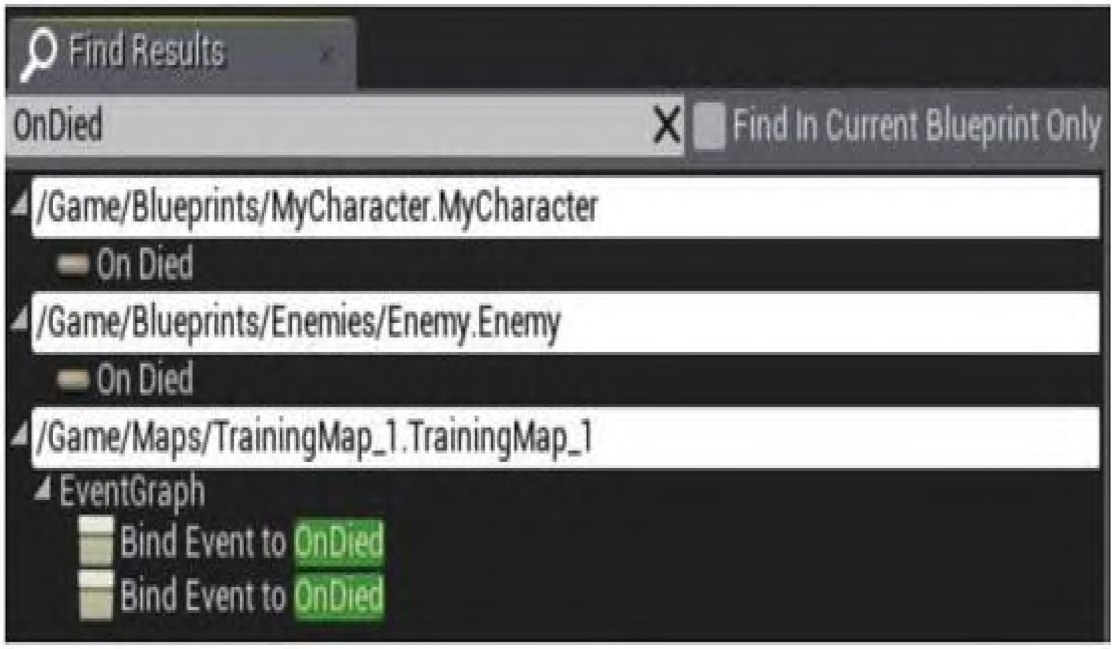



图3-308

•快捷菜单中只显示可连接到拖出引脚的节点（如从一盏灯拖出时，将显示与灯相关的函数）。尝试寻找在变量或组件上可用的所有潜在功能时，只需通过快捷菜单浏览查找即可，如图3-309所示。

•无论是在快捷菜单还是调色板中，搜索条均十分实用。函数上拥有许多关键词标记，即使你不知道函数的名称，也可轻松找到！如图3-310所示。

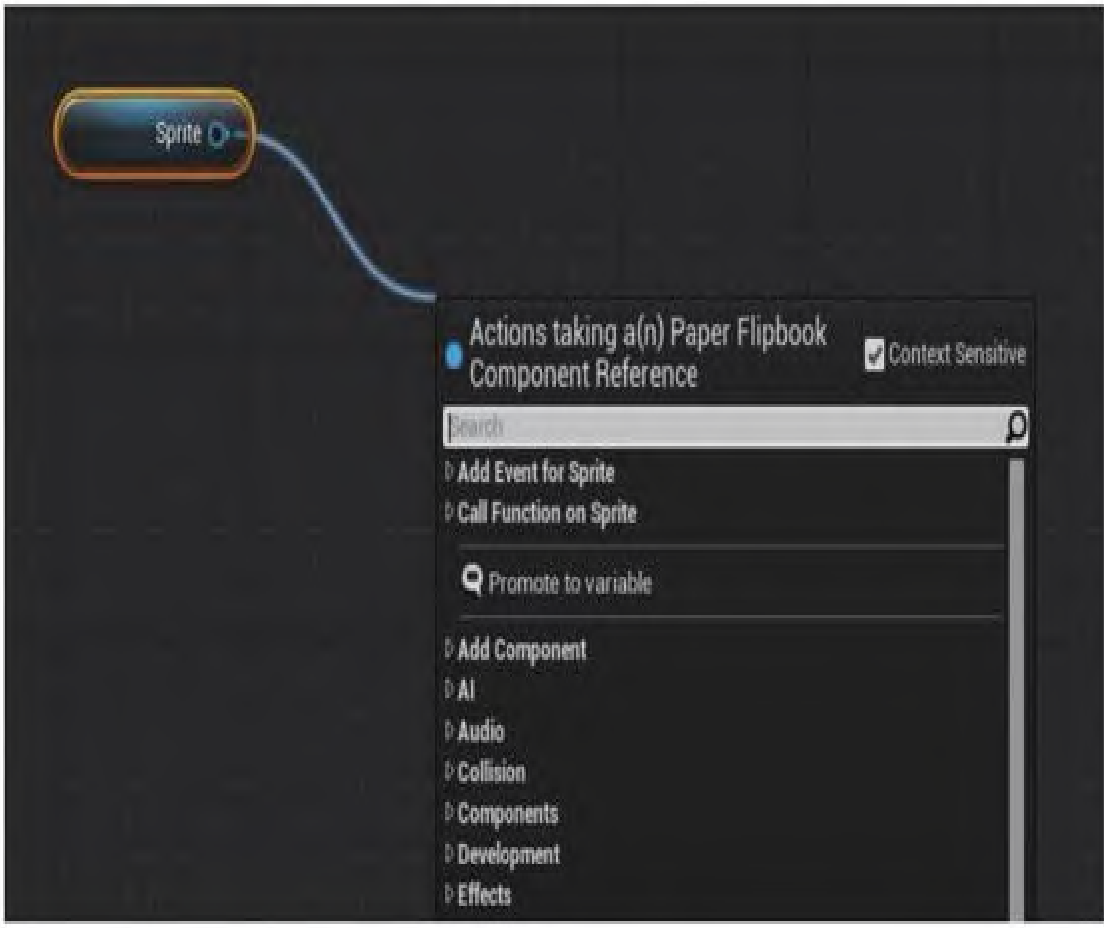

Sprite
Actions taking a(n) Paper Flipbook Component Reference
Context Sensitive
Search
Add Event for Sprite
Call Function on Sprite
Promote to variable
Add Component
AI
Audio
Collision
Components
Development
Effects

图3-309

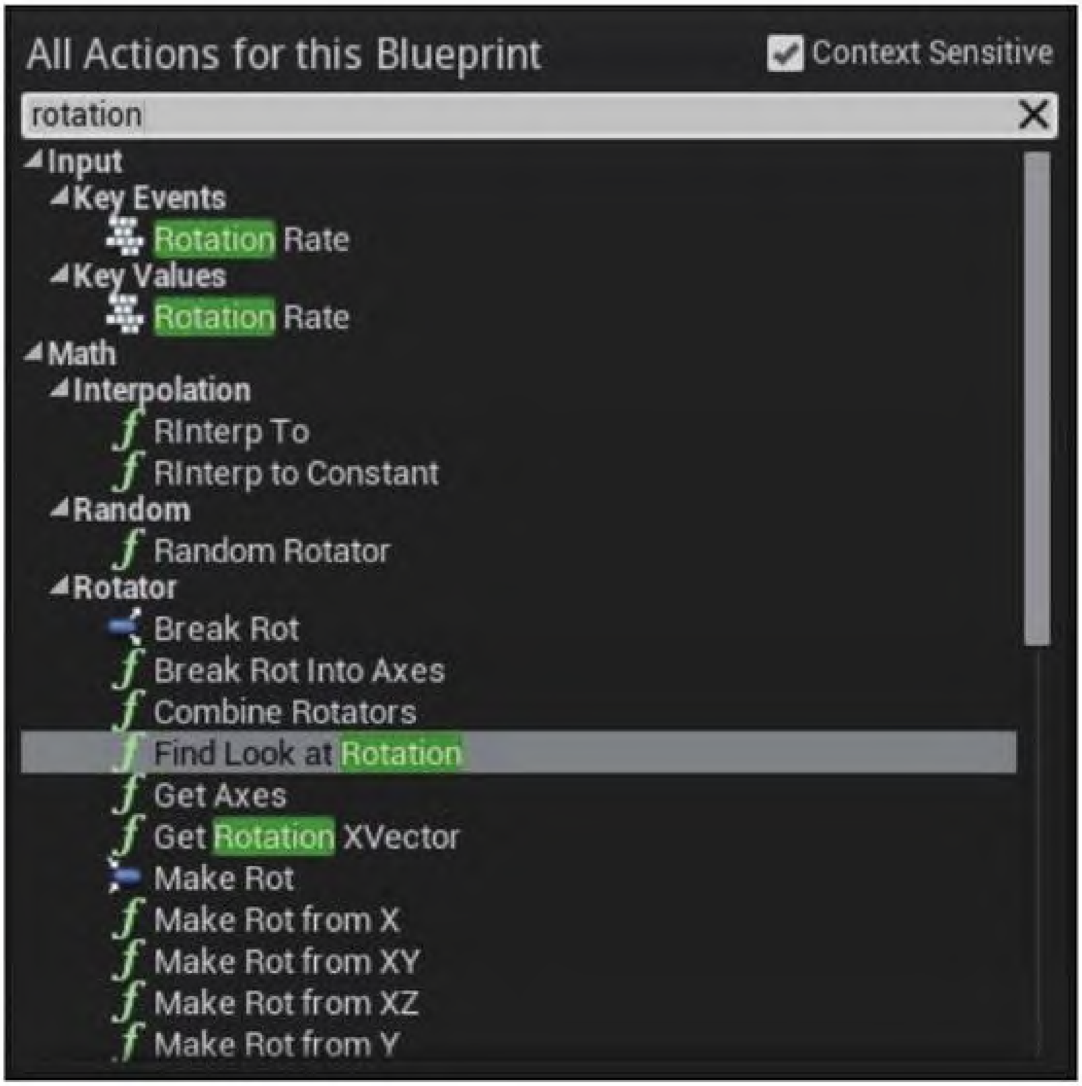



图3-310

### 3.4.5.2 设置图表

每个人的图表设置偏好均不相同，以下提示可助你对内容进行组织，易于使用。

•从最初便保持整齐！在编写代码之后再进行清理便会复杂许多。

•如在一个图表中使用相同节点集超过两次，可考虑将其设为函数或宏以便重复使用。

•通过节点上下文进行堆栈可使内容在空间上更加紧凑。例如，你拥有一个灯光的引用，然后访问其点光源组件，即可将这两个节点相互进行堆栈，将它们视作单个块，如图3-311所示。

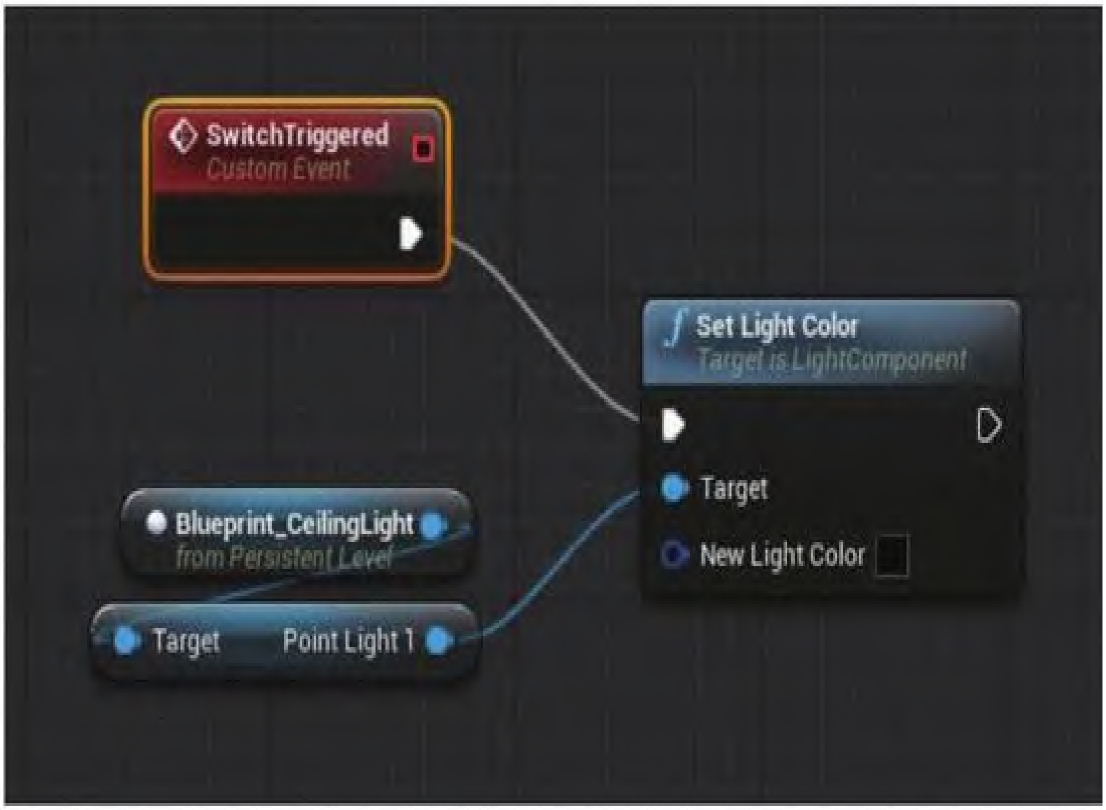



图3-311

• 多使用注释！此外还可变更注释框的颜色，以便识别图表的部分。

第4章

# 蓝图通信

使用蓝图时，如需在蓝图之间传递或共享信息，就需要使用一种形式的蓝图通信，也可根据需求使用数种不同类型的通信。此章讲述最常用的方法和使用情况范例。

4.1

# 蓝图通信用法

何时使用：

以下范例讲述何时使用直接蓝图通信：

•关卡中存在两个Actor，需要进行相互之间的通信。

•与关卡中的一个开关进行互动后打开特定的门或开启特定的灯（每个均为单独的蓝图）。

**1.事件分配器**

事件分配器适用于告知其他“正在倾听的”蓝图已发生事件。事件发生时，正在倾听的蓝图便会作出反应，并相互独立地执行预期的操作，如图4-1所示。

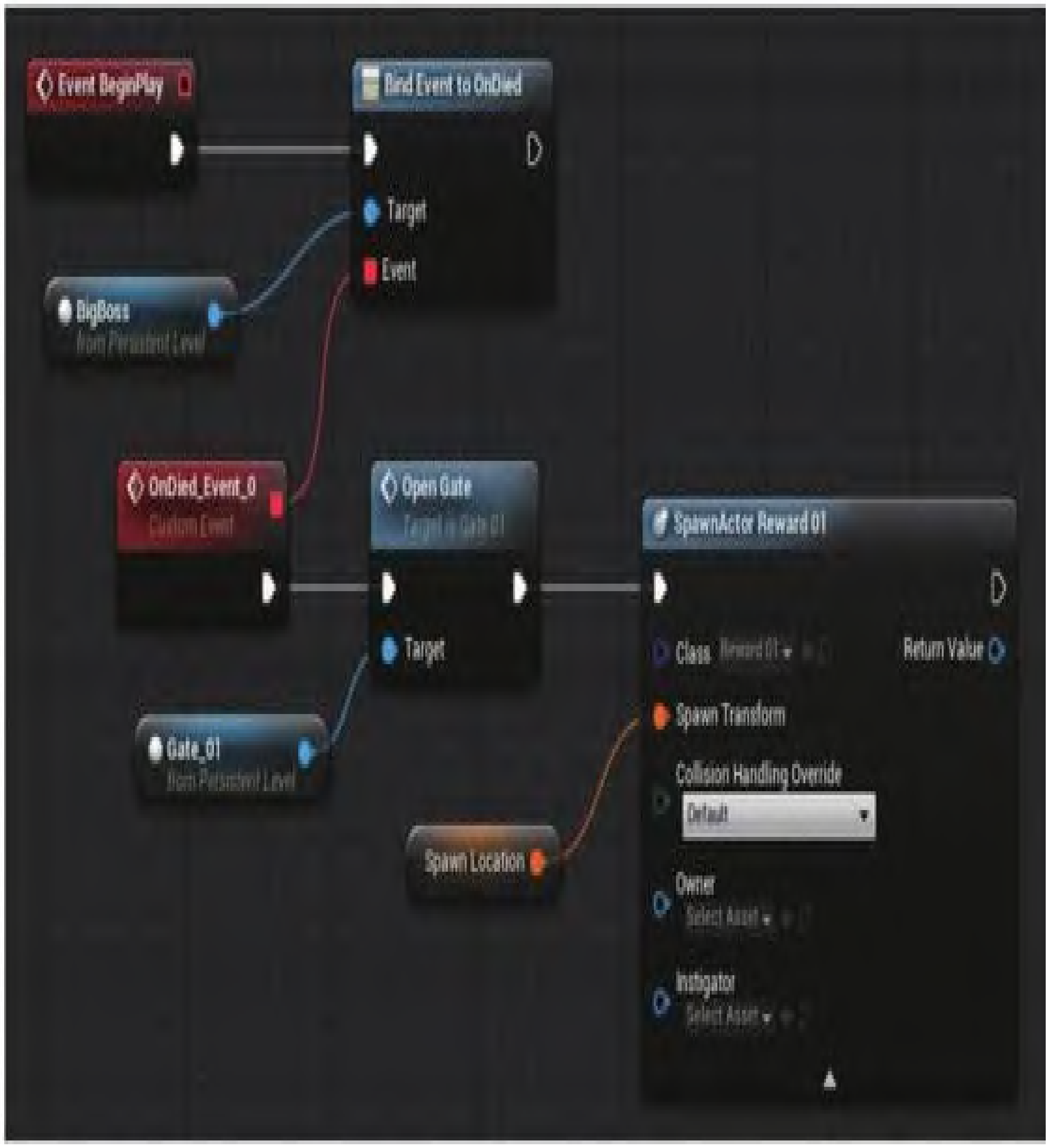



图4-1

例如，游戏中有一个Boss，它被消灭时将会调用“OnDied”事件分配器。可将“OnDied”事件绑定在任意数量的其他蓝图中，如角色（作出庆祝动作）、关卡中的门（会打开）或HUD（显示UI信息）。所有这些蓝图均在收到Boss被消灭的“OnDied”通知后执行。

何时使用？事件分配器使用时机的例子：

•需要从角色蓝图到关卡蓝图进行通信。

◊玩家角色升级，需要开放之前锁定的区域。

◊玩家角色按下行动按钮，对关卡执行某种操作。

•生成的Actor执行某种操作时触发事件。

◊生成一个Boss，Boss被消灭时触发事件，在世界场景中生成一个奖励。

◊在关卡中生成一个道具（武器、回复剂等）并在道具被拾起时告知道具和角色。

**2.蓝图接口**

蓝图接口（简称接口）是实现与多种类型对象（均共享特定功能）形成互动的一般方法。例如汽车和树是完全不同的两种对象，但拥有一个共同点——它们均可被武器开火击中并遭到破坏，如图4-2所示。

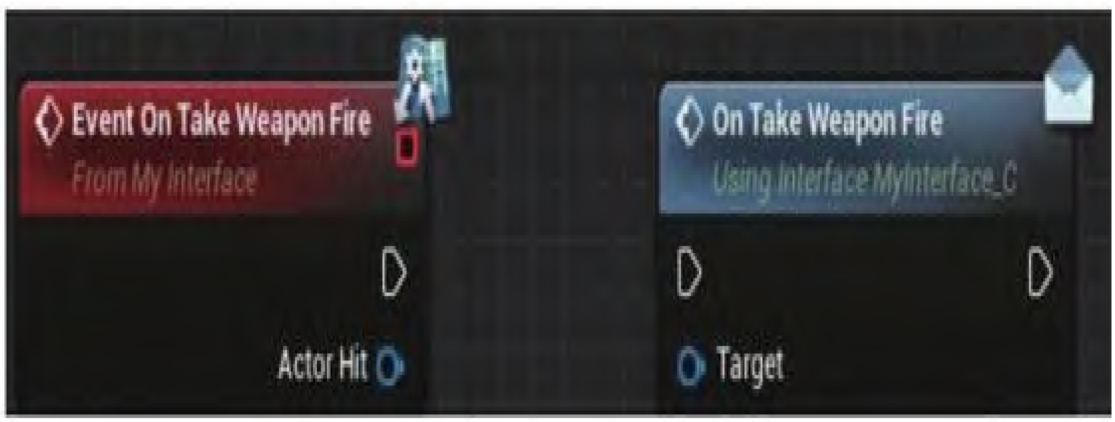



图4-2

创建一个包含OnTakeWeaponFire函数的蓝图接口，让汽车和树应用此蓝图接口后即可将二者视为相同的对象；在任意一个对象被击中时调用OnTakeWeaponFire函数，并使它们出现不同反应。

理解蓝图接口的最佳方式是将其视为一种合约。合约的内容为：“如应用此接口，则须承诺应用这些特定的函数，并在调用时形成反应。”多个蓝图可应用一个接口，但并非所有蓝图均会形成反应（部分蓝图将无视调用）。回到之前武器开火的例子中，墙壁不需要对破坏作出反应，因此它们不会应用带OnTakeWeaponFire函数的接口，并无视调用。

何时使用：

在以下情况实例中，蓝图接口非常适合于通信：

数个蓝图中存在一些相似功能，但在调用后执行不同的效果。

◊玩家拥有一个火焰喷射器，使用时将触发一个名为ElementalDamage的事件。树木蓝图接受ElementalDamage调用并将树木烧毁，而雪人蓝图将把雪人融化。

◊玩家拥有一个“use”按钮，按下按钮可执行开门、关灯、拾起道具等操作。

◊敌人拥有特殊能力，可基于玩家的体力值完成变化，执行不同操作。

3.蓝图投射

蓝图通信的另一种常见形式是使用Cast节点。使用Cast节点时，便是在询问对象“你是否为该对象的特殊版本”。如果是，则进行访问；如否，则忽略请求，如图4-3所示。

图4-3

例如，你创建了一个名为“Flying Character蓝图”的特殊角色蓝图，此蓝图可使玩家角色在游戏中飞行。你可使用Get Player Character节点访问玩家角色的移动组件并通过常规方法对玩家产生影响，如设置其位置、旋转等。你无法获得飞行能力，因为Character蓝图并不了解飞行。只有“Flying Character蓝图”了解飞行。在此例中，你需要获取玩家角色和Cast To Flying蓝图，之后才可访问flying函数。

如玩家角色未使用Flying Character蓝图而完全使用不同类型的Character蓝图，则执行Cast To Flying蓝图时会失败，因为并未使用Flying蓝图，所以无法访问并执行飞行命令。

以下是蓝图投射使用时机的几个例子：

•需要访问另一个蓝图的特殊版本。

◊角色走进火焰中，导致体力值耗尽。

▪投射到特殊的角色蓝图，以便访问并变更体力值。

◊角色死亡，需要重新生成。

▪投射到特殊的游戏模式蓝图，执行重新生成脚本。

•需要访问相同类的多个蓝图，并以相同方法进行修改。

◊场景中拥有数盏灯，事件发生时需将它们开启或关闭。

▪投射到灯蓝图并执行函数将灯关闭。

•需要访问一个特殊的子蓝图。

◊存在基于一个动物蓝图（猫、狗、鸟）的数个蓝图，需要访问其中一个动物。

▪投射到猫、狗和鸟，访问其相应的蓝图和特有功能。

4.2

# 蓝图通信项目

## 4.2.1　概述

在这个Blueprint Communication Project（蓝图通信项目）中，我们会创建简单的射击场风格的游戏。在学习项目的过程中，你会了解到如何在关卡和角色间进行信息交流，以及如何传递角色信息到HUD。在项目的最后，你会看到类似于图4-4所示的效果。

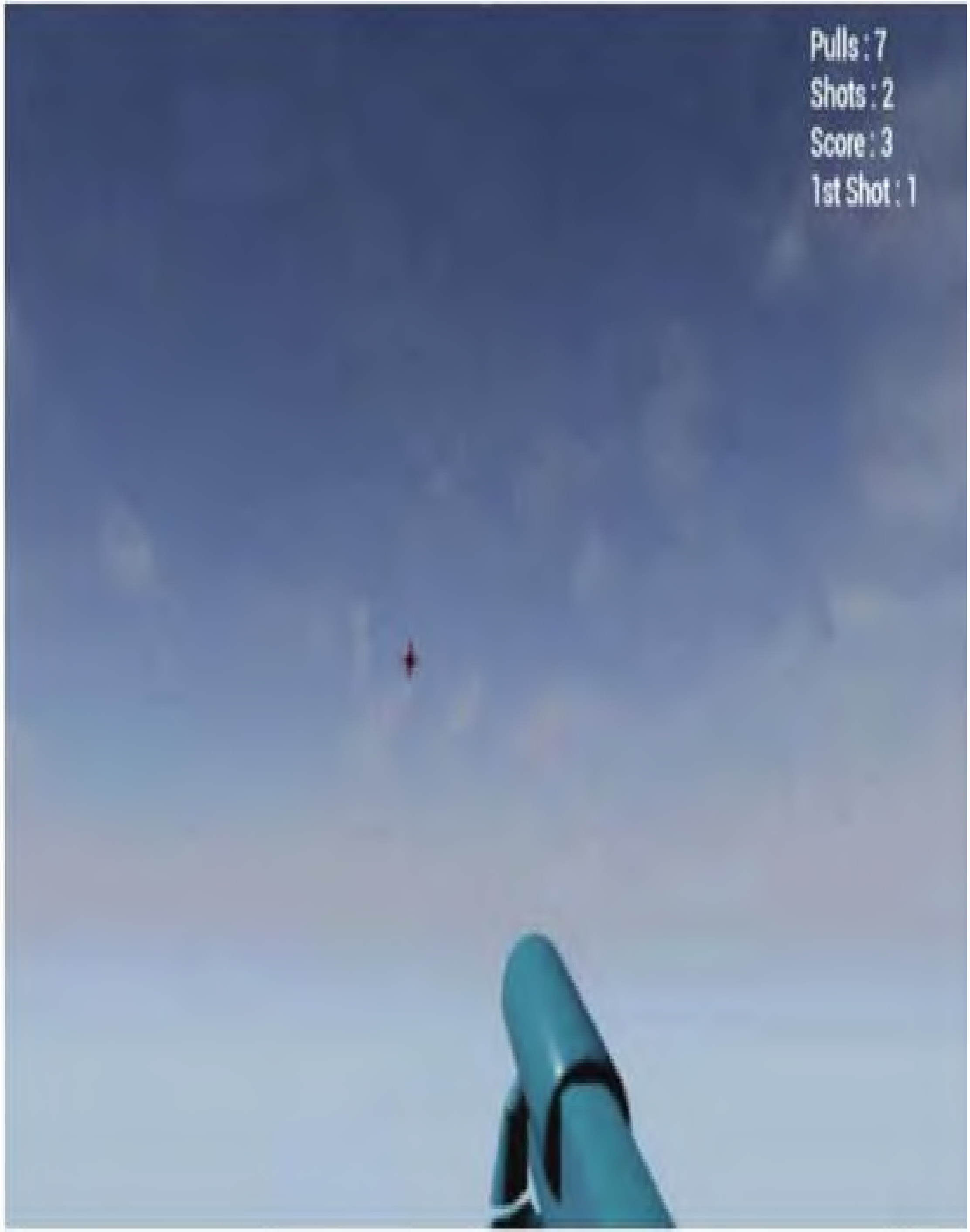



图4-4

在这个简短的游戏示例中，在玩家按下按钮时会从角色蓝图内生成可破坏的静态网格物体。岩石生成的位置和发射的位置是从我们在关卡中放置的生成点处随机选择的，所以我们必须在关卡和角色蓝图间对生成位置进行通信。我们还取入了存储在角色蓝图中的信息（尚未发射的子弹或岩石数量，

每次扣动扳机时剩余的子弹数量以及玩家分数），并将该信息传递到HUD，从而在屏幕上显示该信息。

## 4.2.2　所需资源的设置

在这一节中，我们会进行一些准备工作，并对稍后在本指南中使用的资源进行设置。我们会创建一个可破坏的静态网格物体，放置一些生成位置，并创建基础HUD来对玩家显示游戏状态的信息。

我们在启用了Starter Content（入门内容）的前提下，专为这个教程创建了一个基于蓝图第一人称模板的新项目。如果你不确定如何基于模板创建项目，或者如何启用初学者内容，请查看项目浏览器文档以了解关于设置项目的信息。

在你创建并打开了项目后，请按下Ctrl+N键来创建新关卡，并在弹出窗口时选择默认值。在你创建了新关卡后，请按照以下步骤来开始设置每个所需的资源。

### 4.2.2.1　创建可破坏的静态网格物体

让我们首先创建可射击的岩石，它会在被击中后销毁。

**Step 1**　在内容浏览器中，打开Content/StarterContent/Props文件夹。

**Step 2**　在Props文件夹中，右击SM_Rock并选择Create Destructible Mesh（创建可破坏网格物体），如图4-5所示。

**Step 3**　打开SM_Rock_DM资源，然后单击工具条中的Fracture Mesh（破裂网格物体）按钮，如图4-6所示。如果你没有在预览窗口中看到岩石，请关闭窗口并再打开一次。

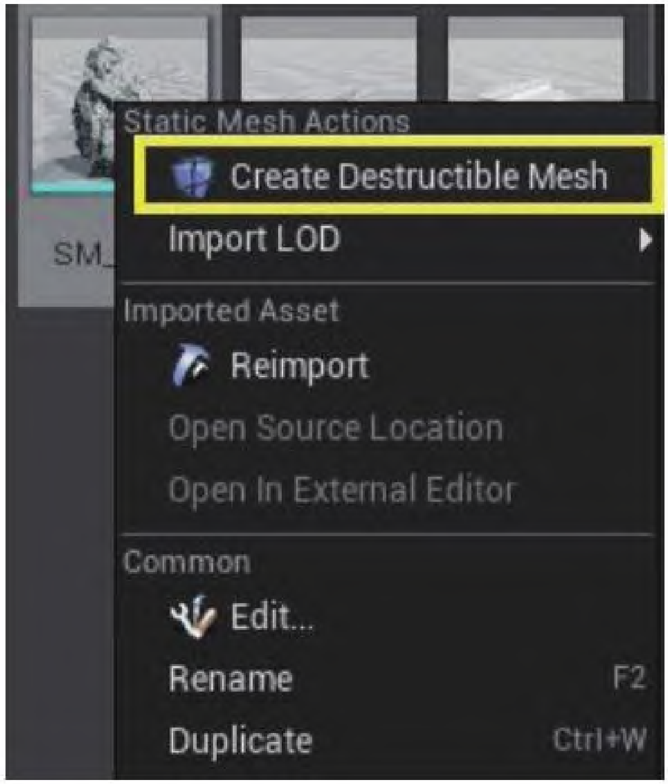



图4-5

图4-6

**Step 4**　保存并关闭SM_Rock_DM资源。

**Step 5**　右击SM_Rock_DM资源并选择Asset Actions/Create Blueprint Using This…（资源操作/使用它来创建蓝图），如图4-7所示。在弹出窗口时，给蓝图起个名称，比如BP_Rock。

**Step 6**　在蓝图打开时，单击Add Component（添加组件）按钮，然后添加一个Projectile Movement（射弹移动）组件，如图4-8所示。

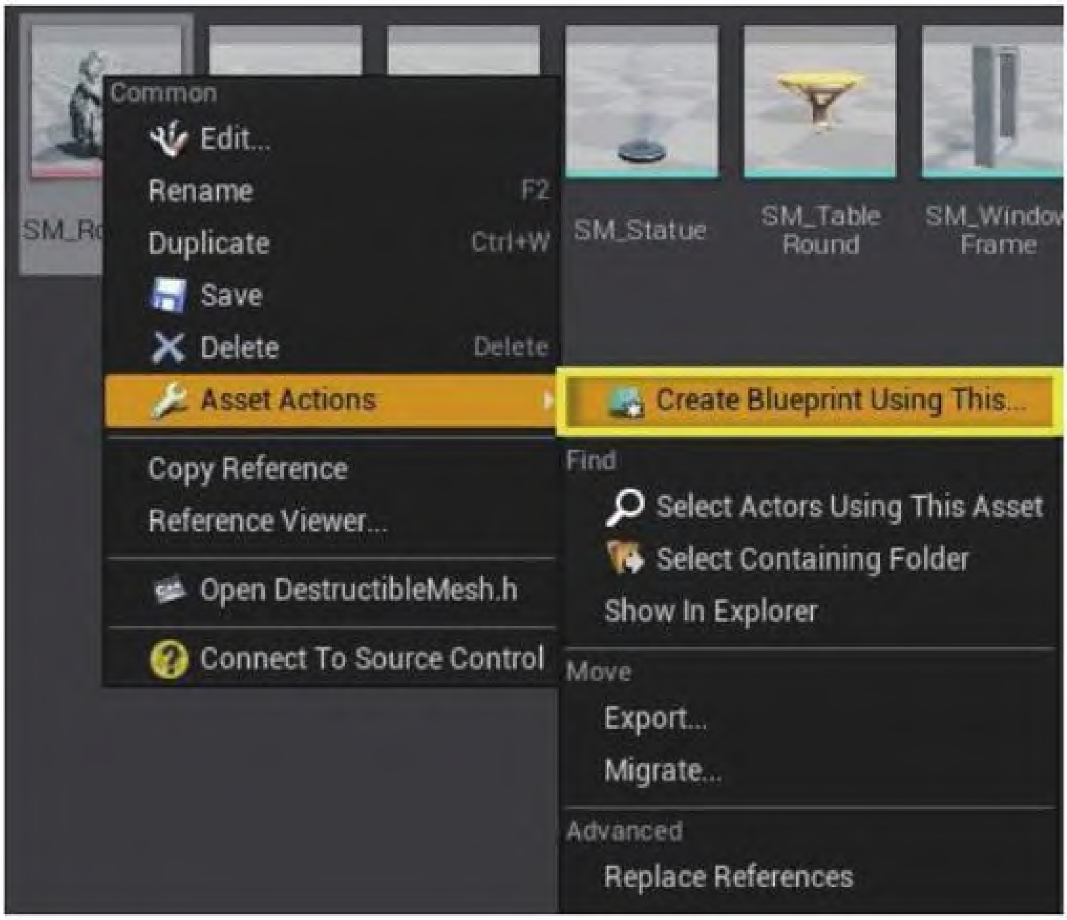



图4-7

图4-8

**Step 7** 单击射弹移动组件，然后在详细信息中设置初始速度、最大速度，并把X的速度设置为1500.00，如图4-9所示。

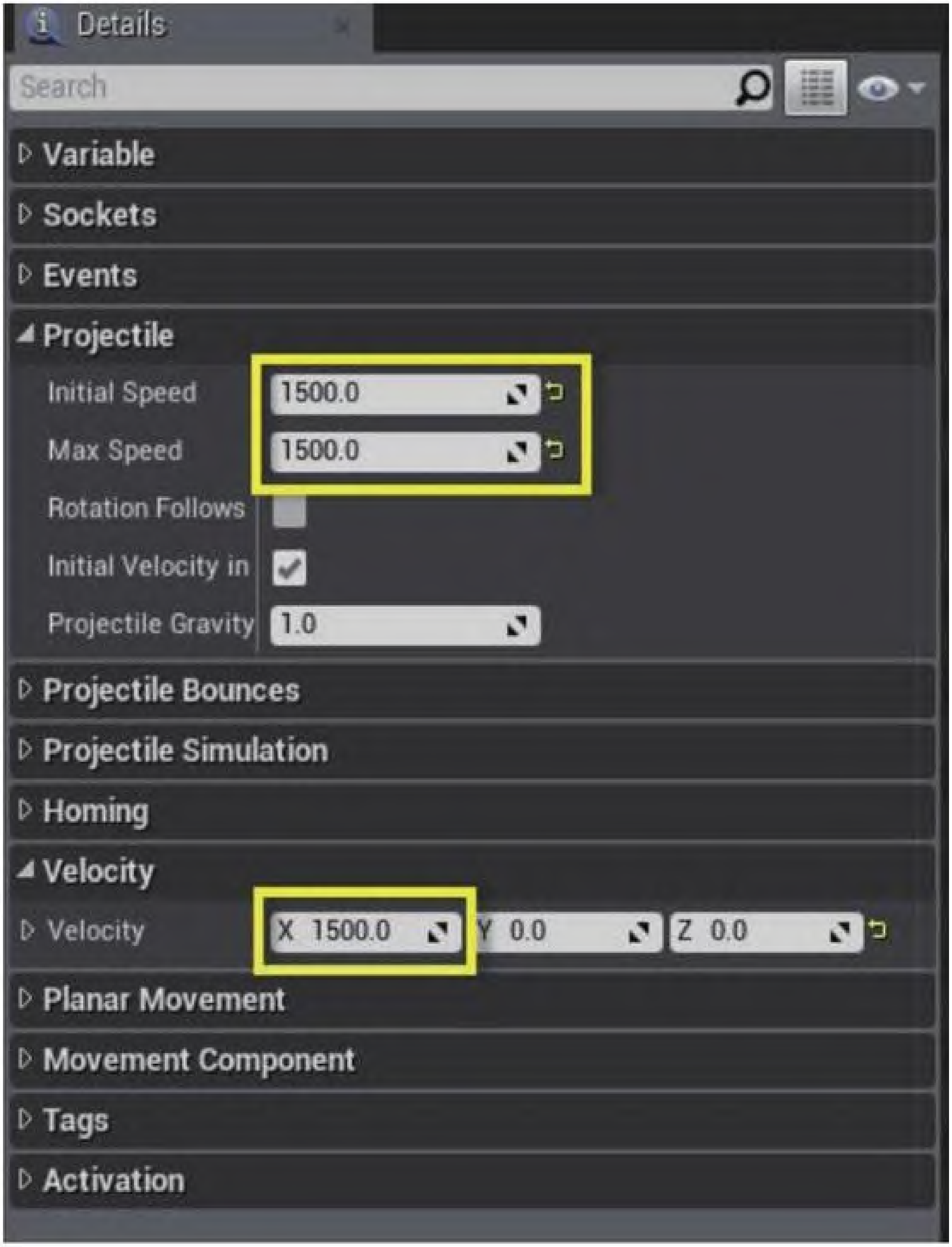



图4-9

**Step 8** 选择可破坏组件，然后在碰撞下的详细信息面板中勾选模拟生成碰撞事件选项，如图4-10所示。

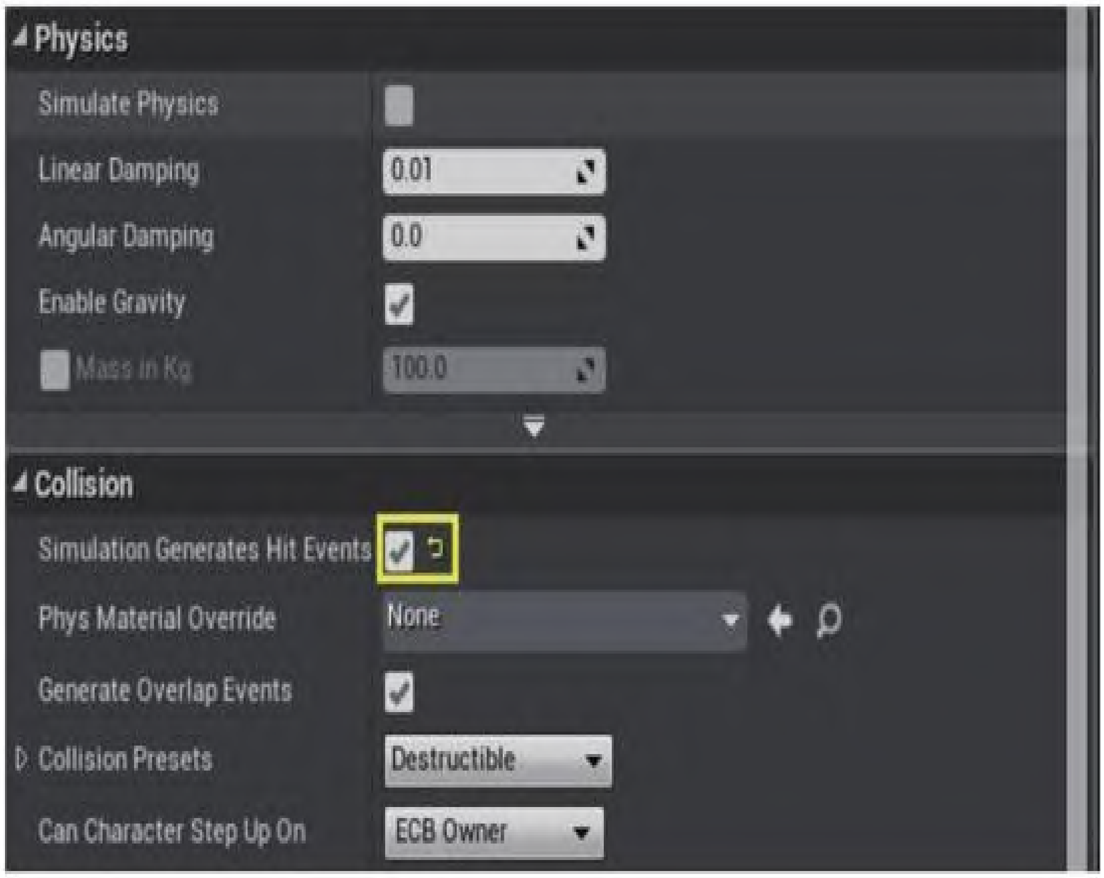



图4-10

Step 9　单击编译和保存按钮，然后关闭蓝图。

#### 4.2.2.2　放置生成点

我们现在会在关卡中创建一些生成点，以启动可射击的岩石。

Step 1　在模式菜单，搜索并添加目标点到关卡中，如图4-11所示。

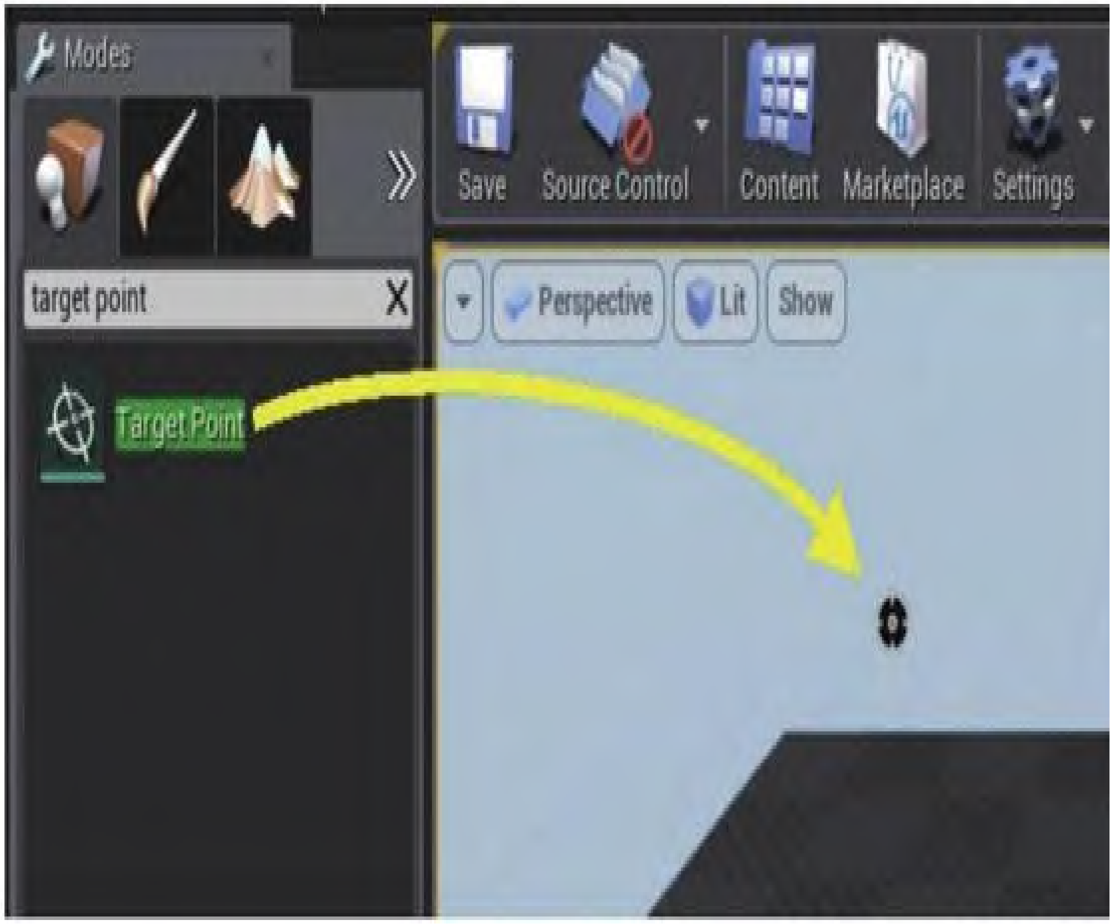



图4-11

**Step 2**　选择关卡中的对象点，并将其变换设置为如下内容，如图4-12所示。

图4-12

X的位置为7.0m（其余为0），而Y的旋转为40.0（其余为0）。

**Step 3** 在关卡中，在选择了目标点后，按下Ctrl+W键来复制它，并按如下方式来设置其详细信息值，如图4-13所示。

图4-13

这样将会把生成点移动到首个生成点的左侧。

**Step 4** 按下Ctrl+W键以再次复制目标点，并按如下方式来设置新目标点的详细信息，如图4-14所示。

现在你在玩家起始位置处有了3个生成点（控制器图标和方向由蓝色箭头来指示），如图4-15所示。

图4-14

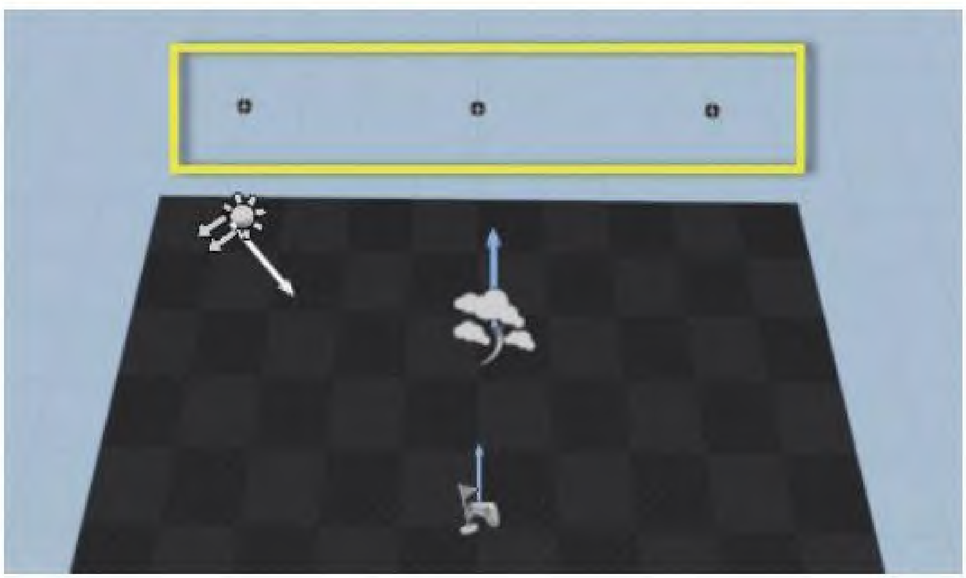


图4-15

#### 4.2.2.3 创建HUD

在设置的最后，需要我们创建基础HUD，从而进行信息通信。

**Step 1** 在内容浏览器的空白处，右击并选择User Interface/Widget Blueprint（用户接口/控件蓝图），如图4-16所示。

控件蓝图指的是用来创建HUD元素、与Unreal Motion Graphics UI（虚幻运动图形UI）共同使用的蓝图类型。

**Step 2** 在提示时，将其命名为HUD并打开。

**Step 3** 在面板下的Palette窗口中，把Vertical Box（垂直框体）拖曳到Hierarchy（层次结构）窗口的CanvasPanel中，如图4-17所示。

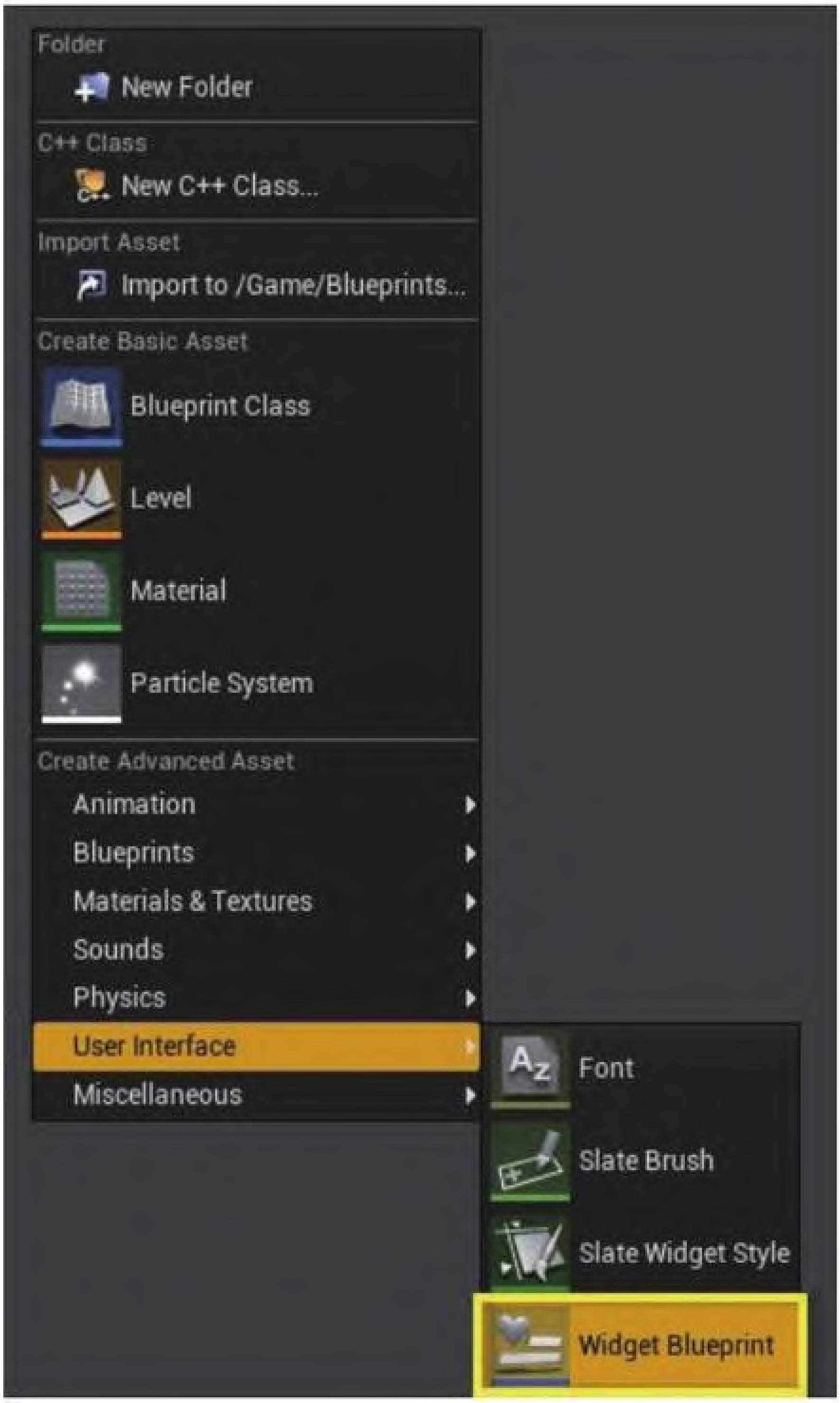

Folder
New Folder
C++ Class
New C++ Class...
Import Asset
Import to /Game/Blueprints...
Create Basic Asset
Blueprint Class
Level
Material
Particle System
Create Advanced Asset
Animation
Blueprints
Materials & Textures
Sounds
Physics
User Interface
Miscellaneous
Font
Slate Brush
Slate Widget Style
Widget Blueprint

图4-16

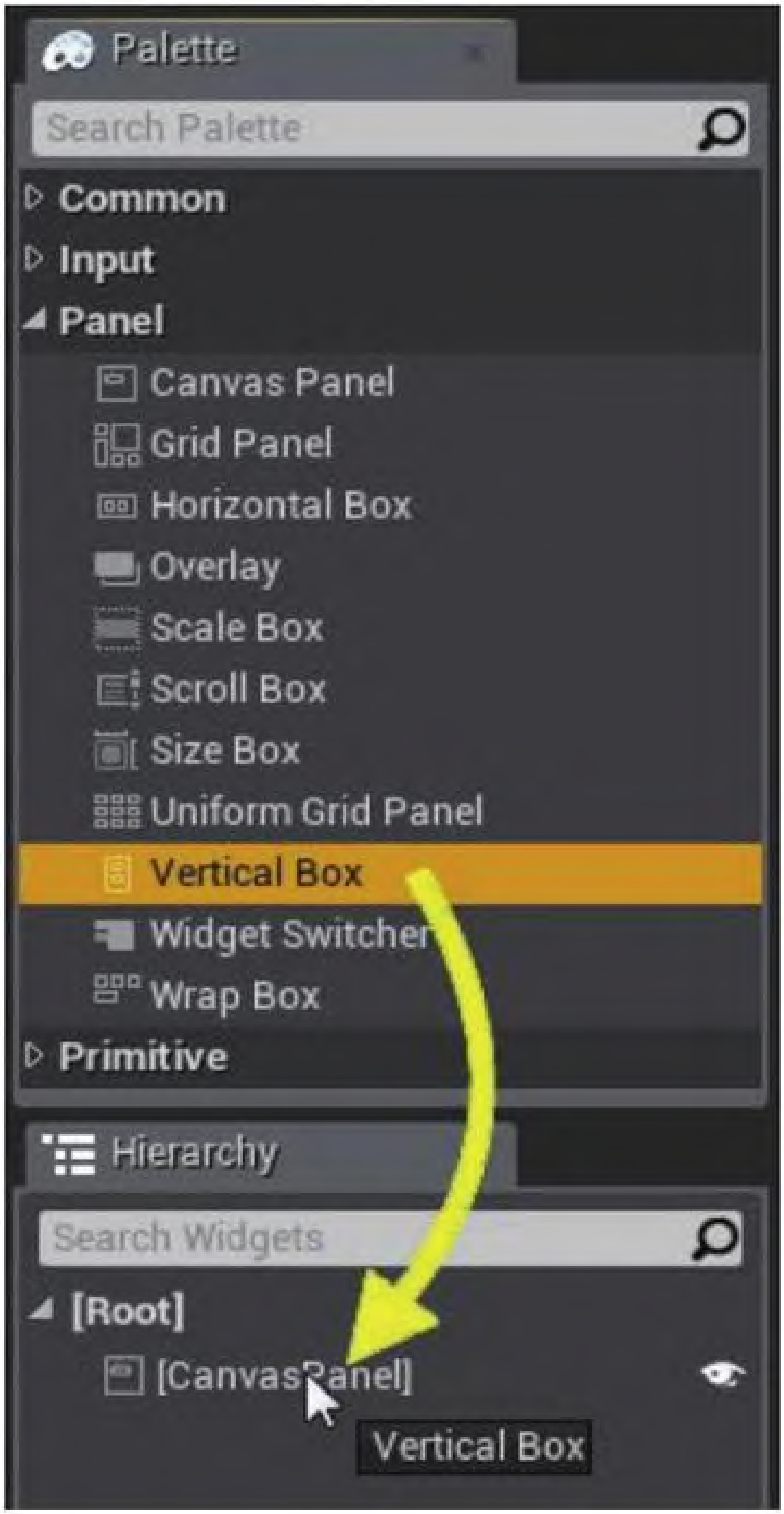

Palette
Search Palette
Common
Input
Panel
Canvas Panel
Grid Panel
Horizontal Box
Overlay
Scale Box
Scroll Box
Size Box
Uniform Grid Panel
Vertical Box
Widget Switcher
Wrap Box
Primitive
Hierarchy
Search Widgets
[Root]
[CanvasPanel]
Vertical Box

图4-17

**Step 4**　另外在面板菜单下，拖动Horizontal Box（水平框体）到Hierarchy（层次结构）窗口中的Vertical Box（垂直框体），如图4-18所示。

**Step 5**　在Palette下的Common（通用）部分，拖动两个Text（文本）控件到Hierarchy（层次结构）窗口中的Horizontal Box（水平框体），如图4-19所示。

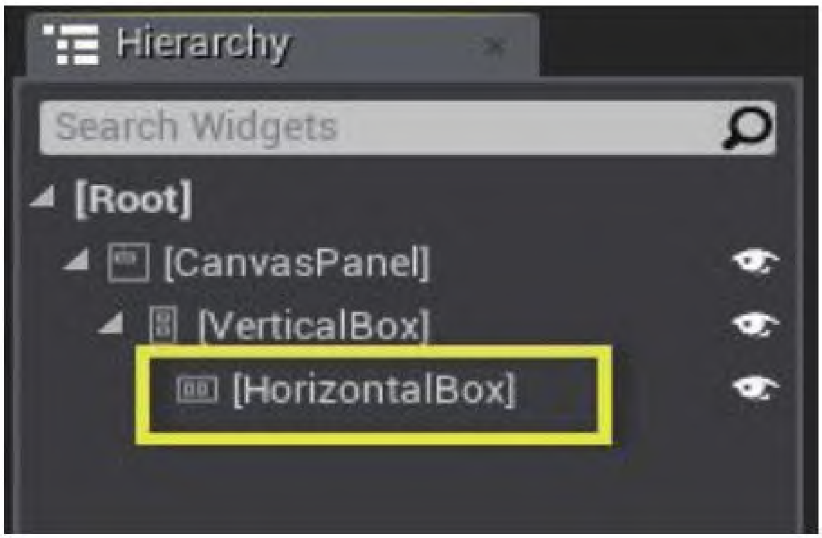



图4-18

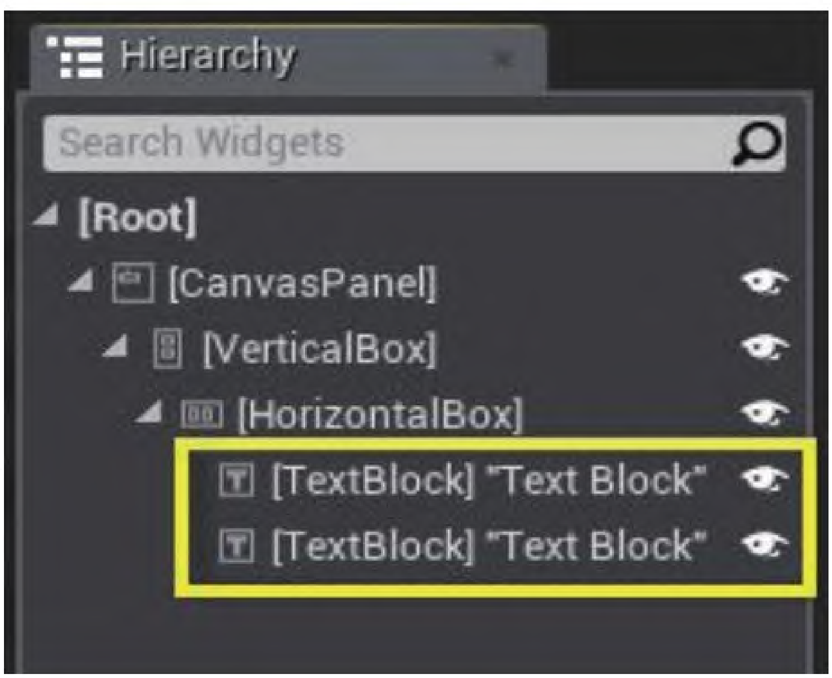



图4-19

**Step 6**　选择Hierarchy（层次结构）窗口中的Vertical Box（垂直框体），然后在详细信息面板中单击Anchors（定位点）并选择以下内容，如图4-20所示。

无论屏幕大小如何，Anchor（定位点）都会锁定指定位置的UI元素。

**Step 7**　在选择了Vertical Box（垂直框体）后，将其移动到屏幕右侧并对其进行缩放，使文本能够显示，如图4-21所示。

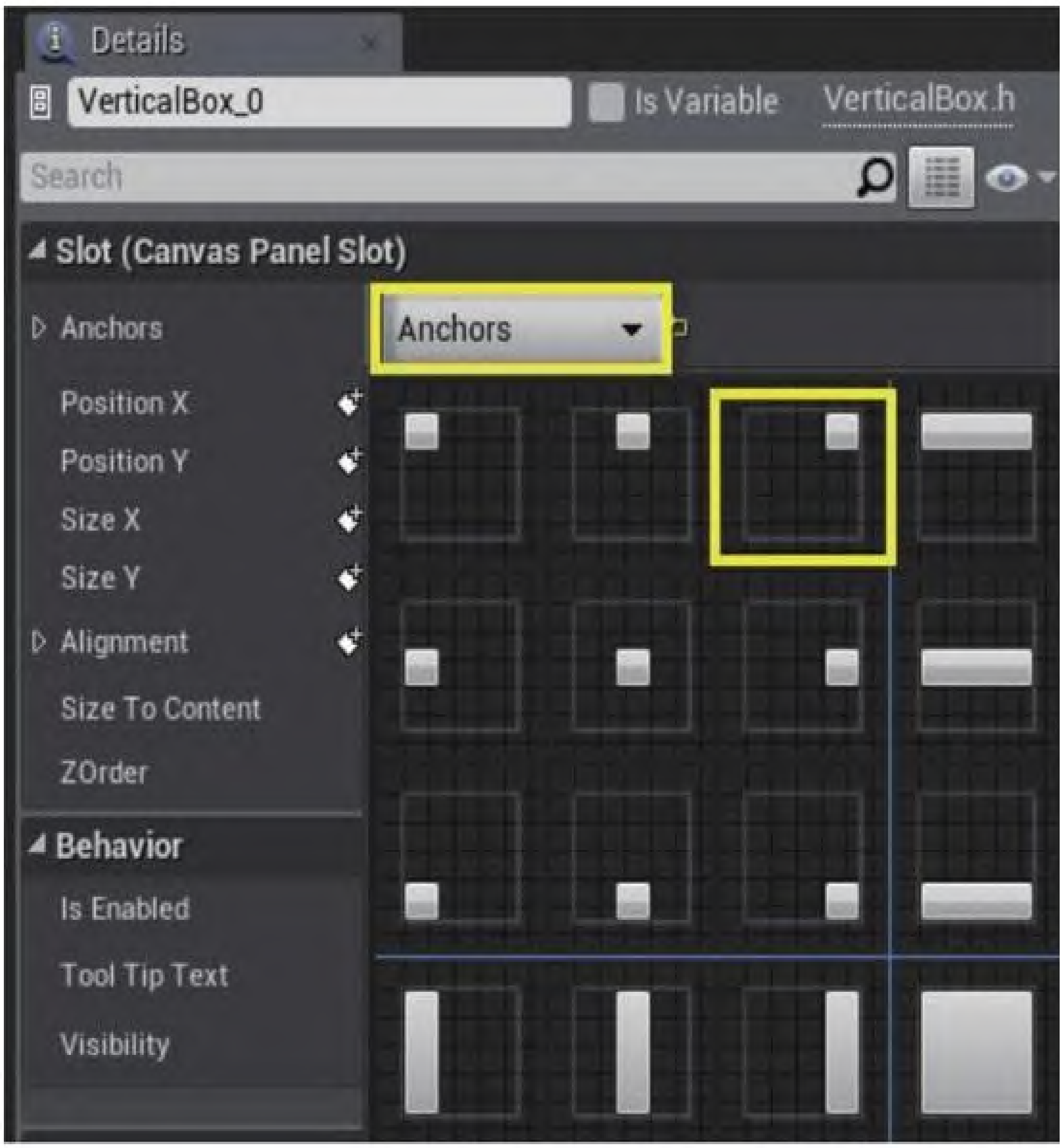



图4-20

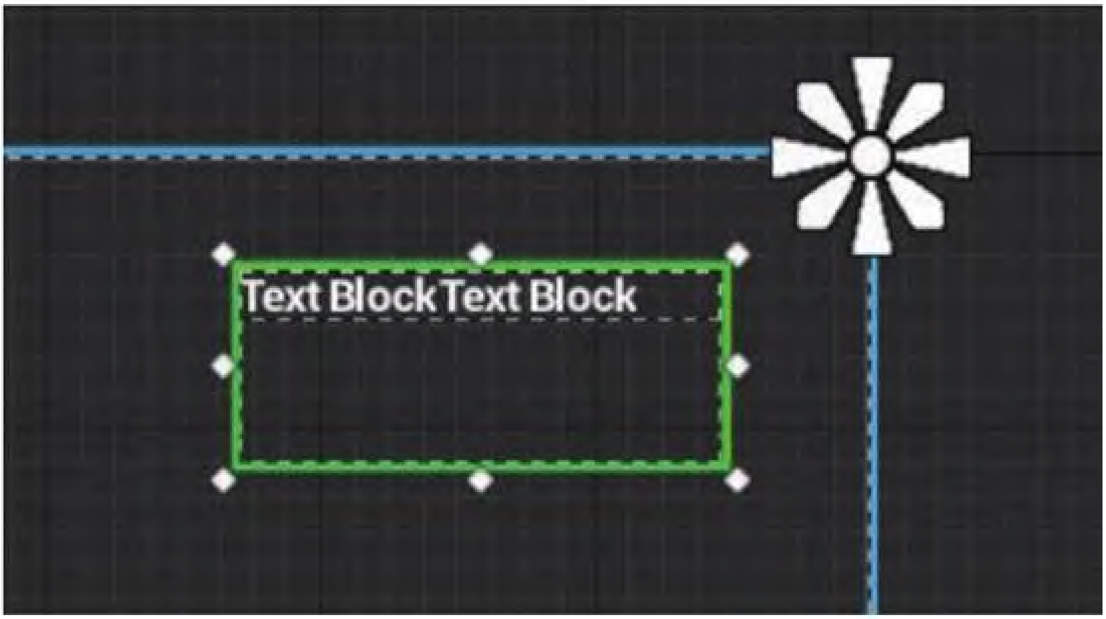



图4-21

**Step 8** 单击框体中显示的最左侧的文本，然后在Text（文本）下的Details（详细信息）面板中输入Pulls：，如图4-22所示。

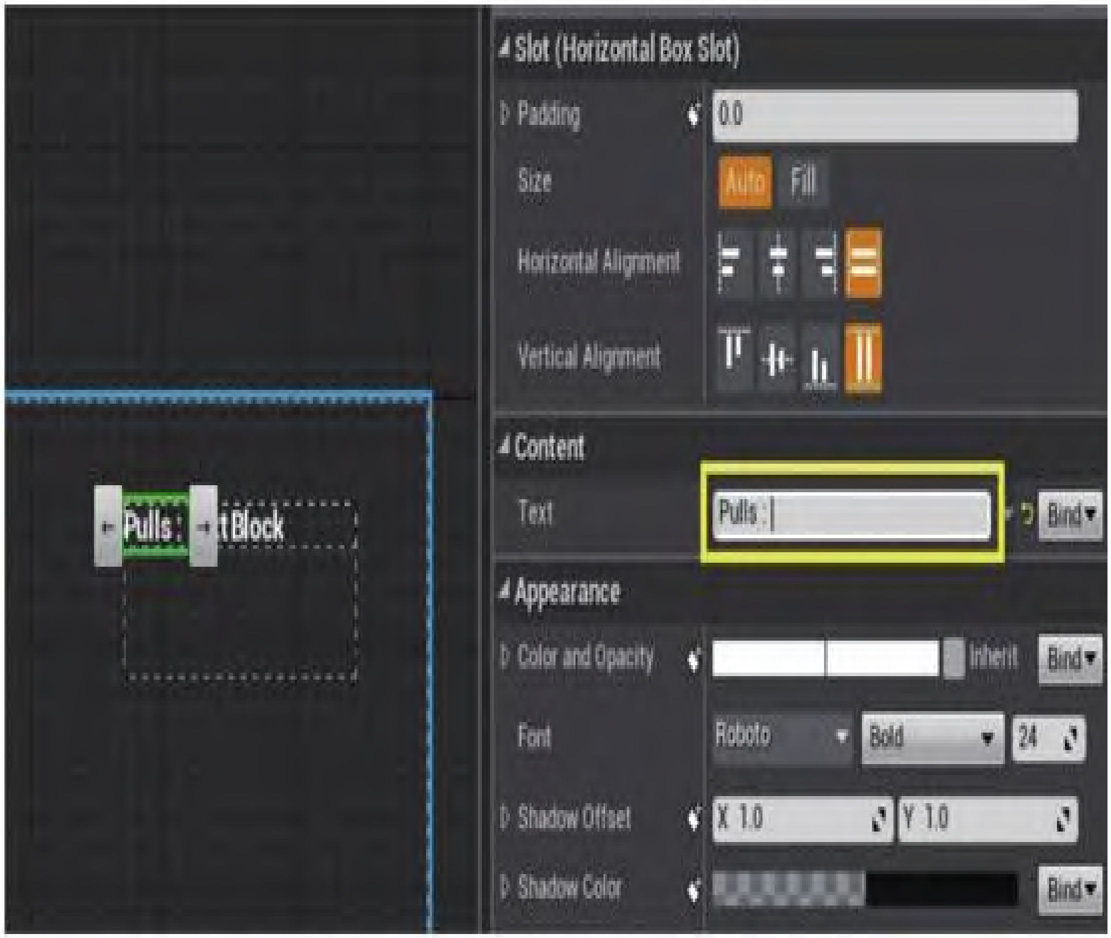



图4-22

**Step 9** 对于其他文本，在Text（文本）域中输入10，如图4-23所示。

图4-23

这个文本就是我们基于游戏状态在稍后进行通信和更新的内容。

**Step 10**　在Hierarchy（层次结构）窗口，右击并复制Horizontal Box（水平框体），然后将其粘贴到Vertical Box（垂直框体）上。

**Step 11**　重复之前的步骤，这样目前在Vertical Box（垂直框体）下会有三个Horizontal Box（水平框体），而文本会如下方那样显示，如图4-24所示。

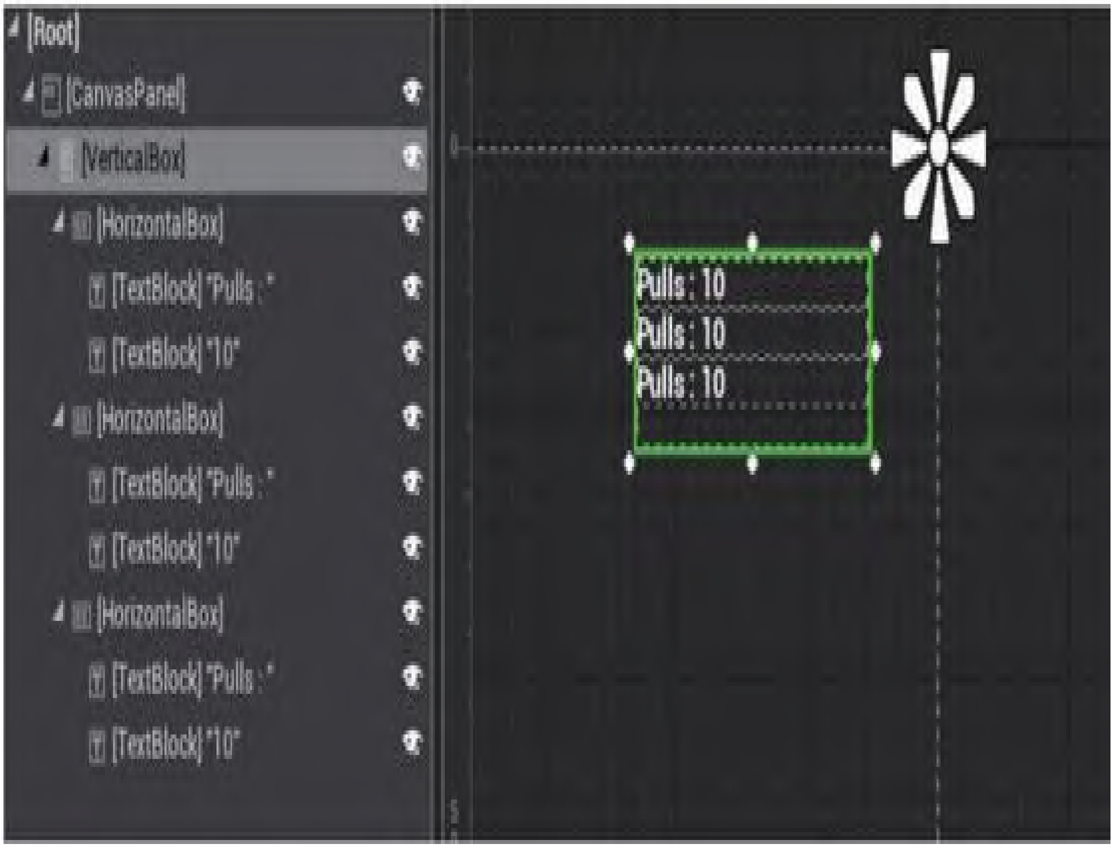



图4-24

**Step 12**　更新其他文本域内容为Shots（设置为3）和Score（设置为0），并单击编译按钮，如图4-25所示。

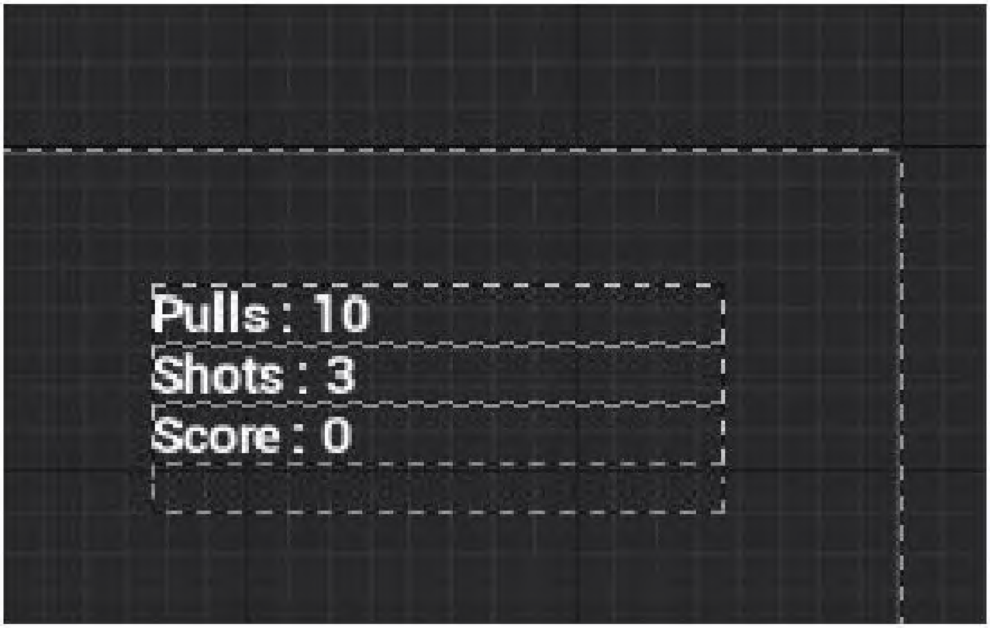



图4-25

**Step 13** 编译并保存你的蓝图，然后关闭HUD Widget Blueprint（HUD控件蓝图）。

现在我们的资源设置就完成了。在下一步中，我们会对Character Blueprint（角色蓝图）进行一些小小的变更，从而确保游戏中的可玩角色进行了正确设置。

## 4.2.3 对所需角色的设置

在这一节中，我们会通过添加稍后在本指南中传递到HUD的变量来完成对角色蓝图的设置。我们还会添加一些条件来确定玩家射击发射的岩石的次数。

### 4.2.3.1 角色设置

让我们开始设置角色蓝图。

**Step 1** 在Content/FirstPersonBP/Blueprints下的Content Browser（内容浏览器），打开First Person Character（第一人称角色）蓝图，如图4-26所示。

**Step 2** 在图表中，删除Stick Input，Movement Input和Jump部分，这样你看到的就如图4-27所示。

Content
FirstPersonBP
Blueprints
Filters
Search Blueprints
FirstPerson Character
FirstPerson GameMode
FirstPerson HUD
FirstPerson Projectile

图4-26

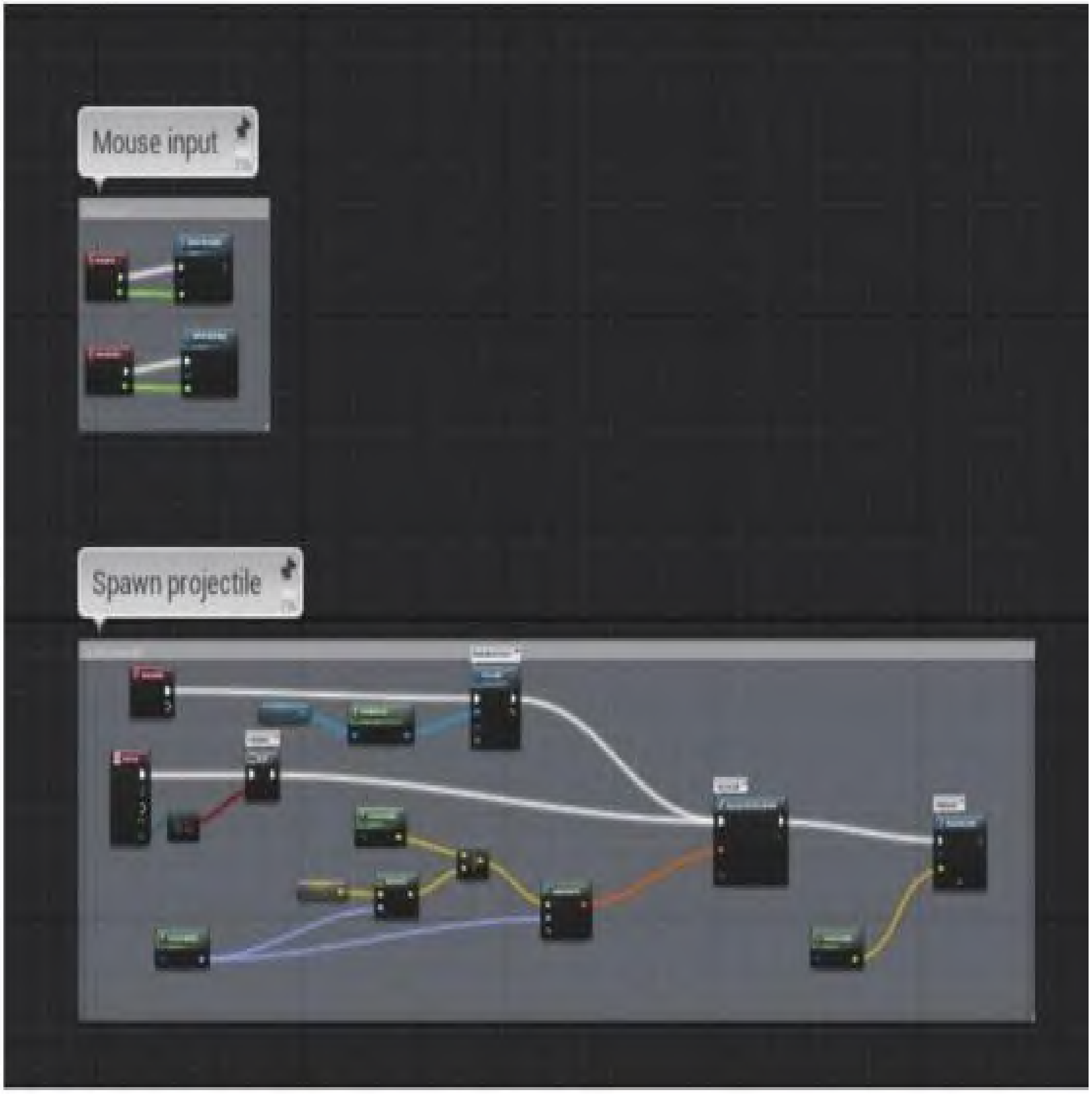



图4-27

对于示例游戏来说，我们只需让玩家转身和射击，因此我们移除了其他脚本来让蓝图看起来更简洁。

**Step 3**　在图表中右击并搜索和添加Event Begin Play（事件开始播放）节点，如图4-28所示。

**Step 4**　把Event Being Play节点拉出来，然后搜索并添加Create Widget（创建控件）节点，如图4-29所示。

图4-28

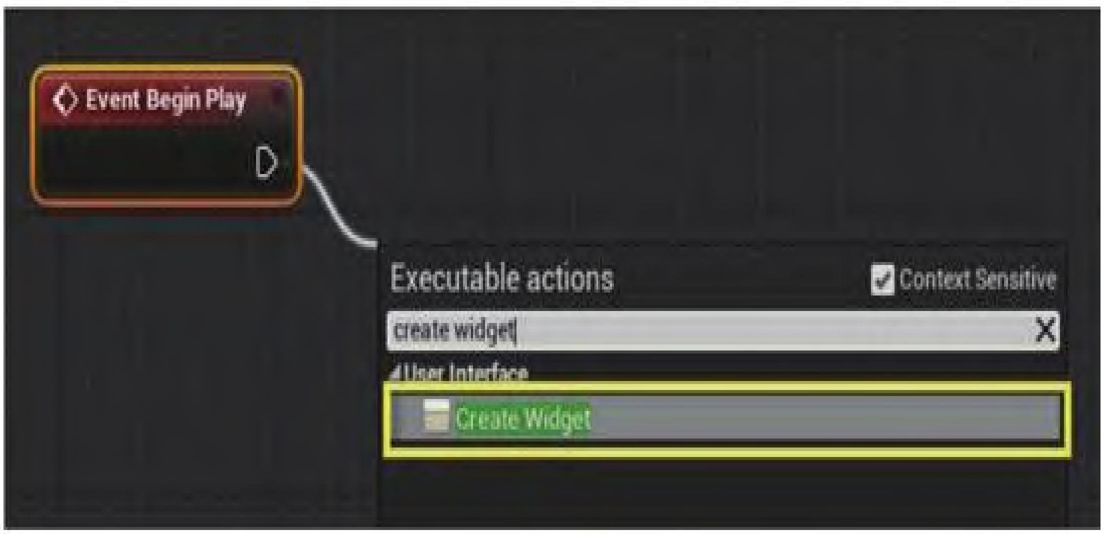



图4-29

我们会在开始玩游戏时调用HUD控件蓝图并显示它。

**Step 5** 在Class（类）的Create Widget（创建控件）节点上，选择HUD，然后关闭Return Value（返回值），使用Add to Viewport（添加到视口）节点，如图4-30所示。

现在会显示出HUD，我们需要创建一些变量来传递给它。让我们现在进行此操作。

**Step 6** 在MyBlueprint窗口，单击Add Variable（添加变量）按钮，如图4-31所示。

**Step 7** 选择新变量，然后在详细信息面板中，将其重命名为isActive并确认其为Boolean（布尔）类型，如图4-32所示。

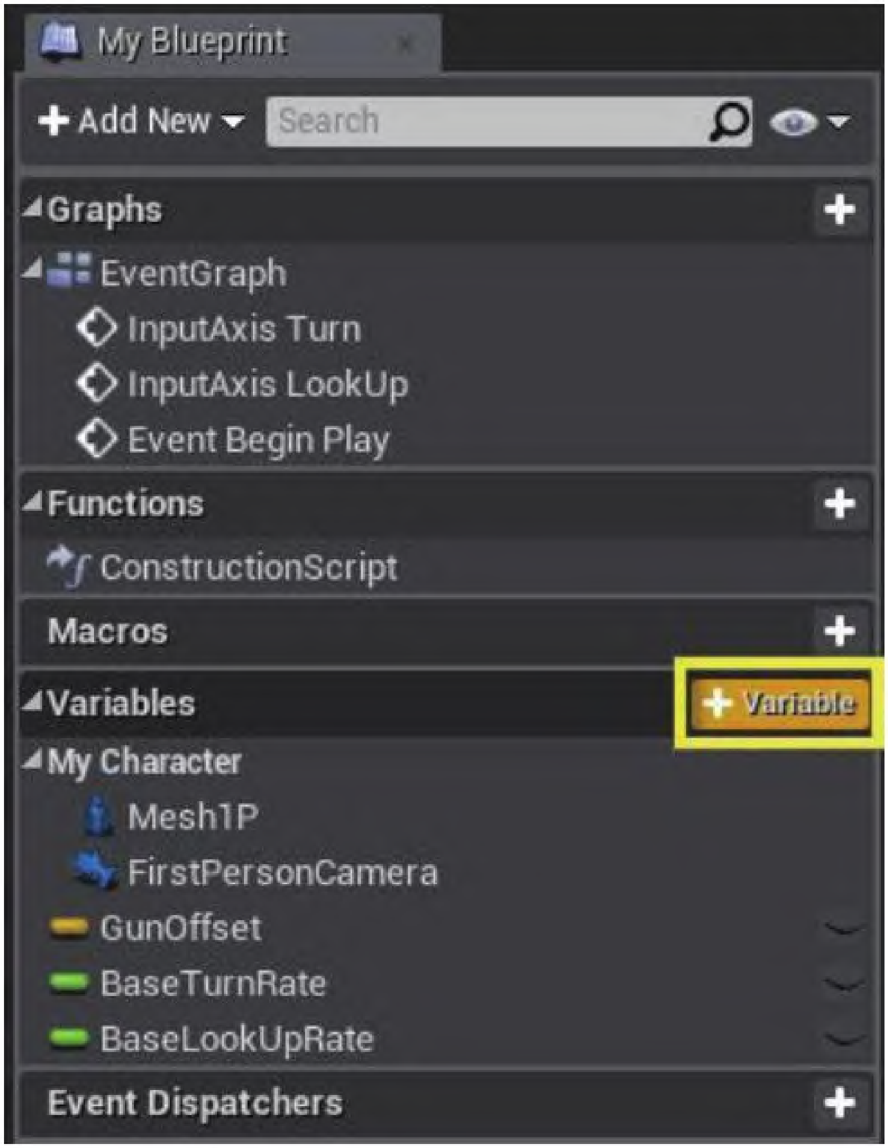

My Blueprint
Add New
Search
Graphs
EventGraph
InputAxis Turn
InputAxis LookUp
Event Begin Play
Functions
ConstructionScript
Macros
Variables
Variable
My Character
Mesh1P
FirstPersonCamera
GunOffset
BaseTurnRate
BaseLookUpRate
Event Dispatchers

图4-31

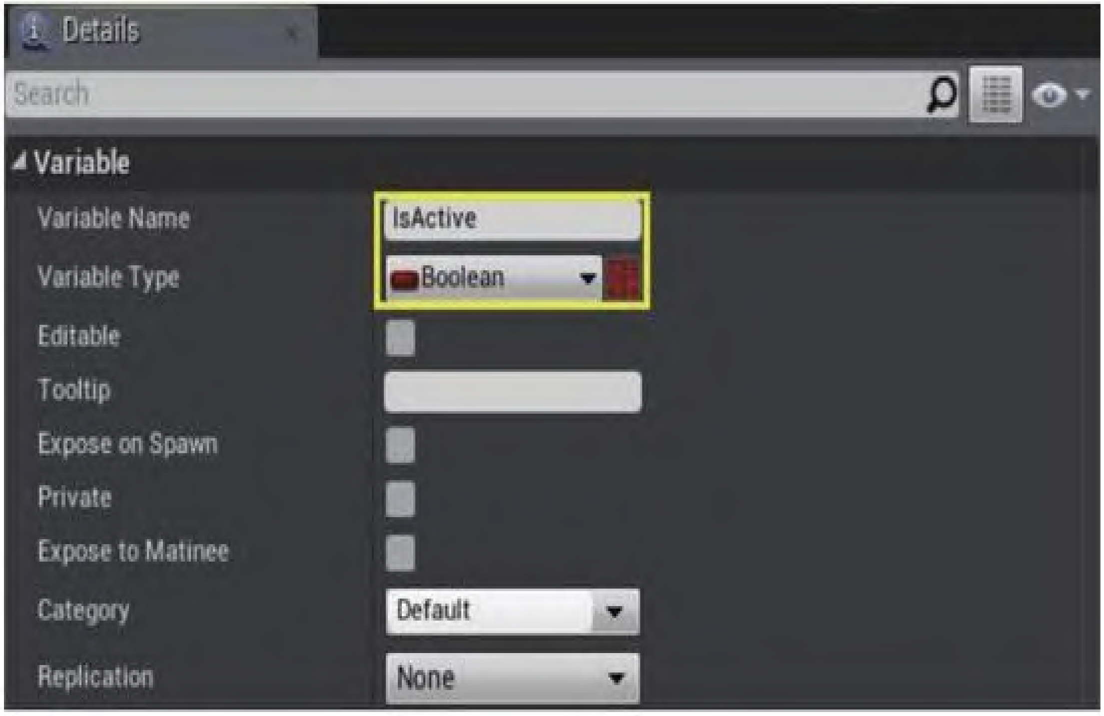



图4-32

**Step 8**　创建另一个变量，将其命名为SpawnLocation，并设置其类型为Transform，如图4-33所示。

这个变量将被用于告知岩石生成的位置并从关卡处进行更新。

**Step 9**　创建另一个变量，将其命名为Pulls，并设置其类型为Integer，如图4-34所示。

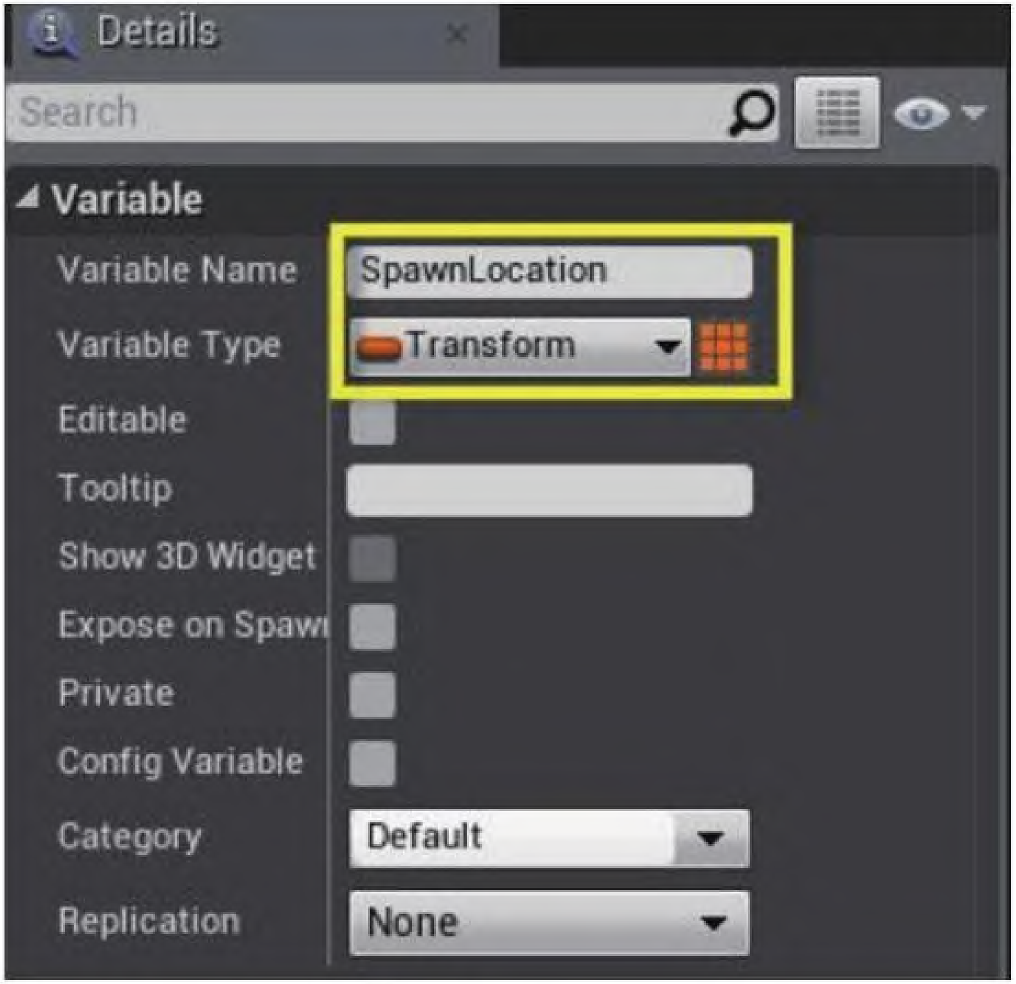

Details
Search
Variable
Variable Name
SpawnLocation
Variable Type
Transform
Editable
Tooltip
Show 3D Widget
Expose on Spawn
Private
Config Variable
Category
Default
Replication
None

图4-33

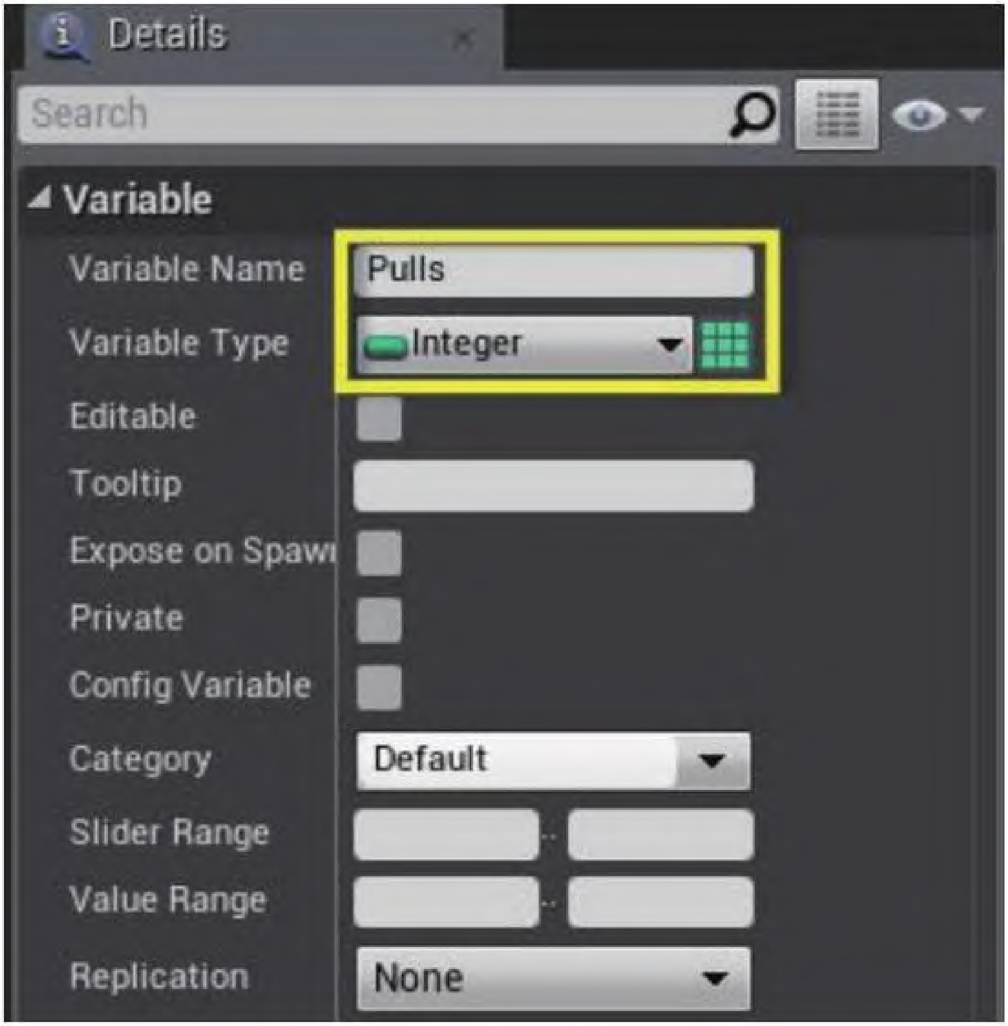



图4-34

这个变量将会通过按钮来保存玩家可以生成的岩石数量。

**Step 10** 再创建两个Integer（整数）变量，一个命名为Shots，另一个命名为Score，如图4-35所示。

你添加的变量应该看起来和图上方的内容类似。

**Step 11** 单击Compile（编译）按钮，然后在Pulls的详细信息面板中，将其默认值设置为10，如图4-36所示。

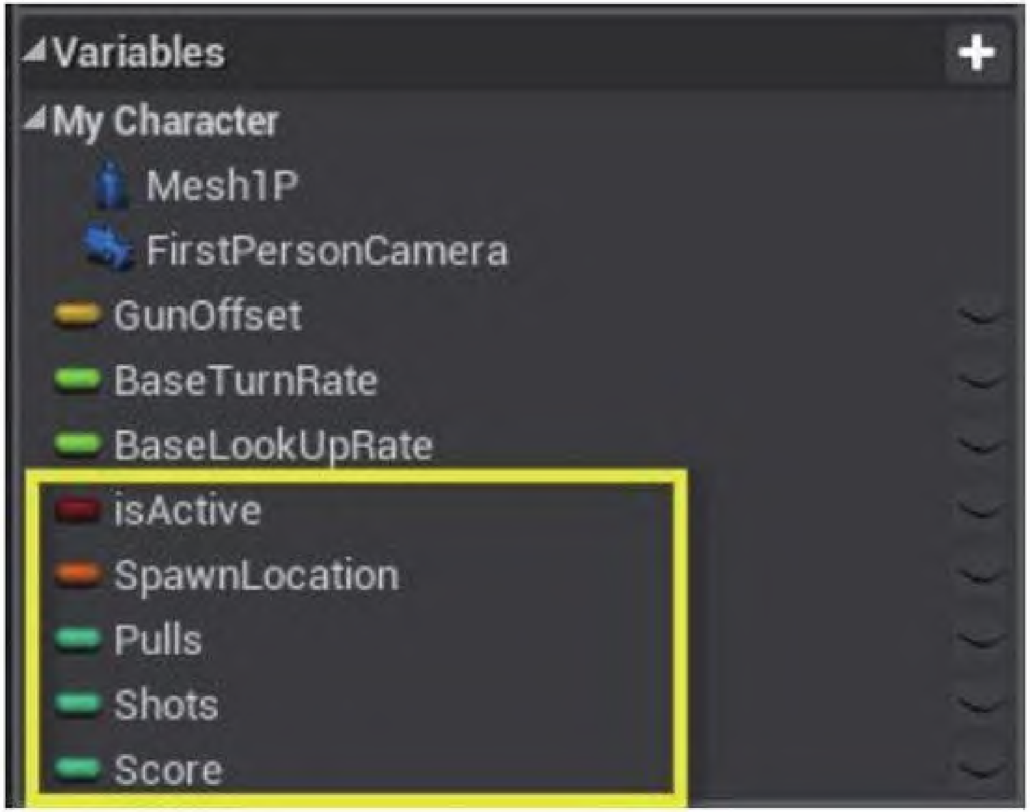

Variables
My Character
Mesh1P
FirstPersonCamera
GunOffset
BaseTurnRate
BaseLookUpRate
isActive
SpawnLocation
Pulls
Shots
Score

图4-35

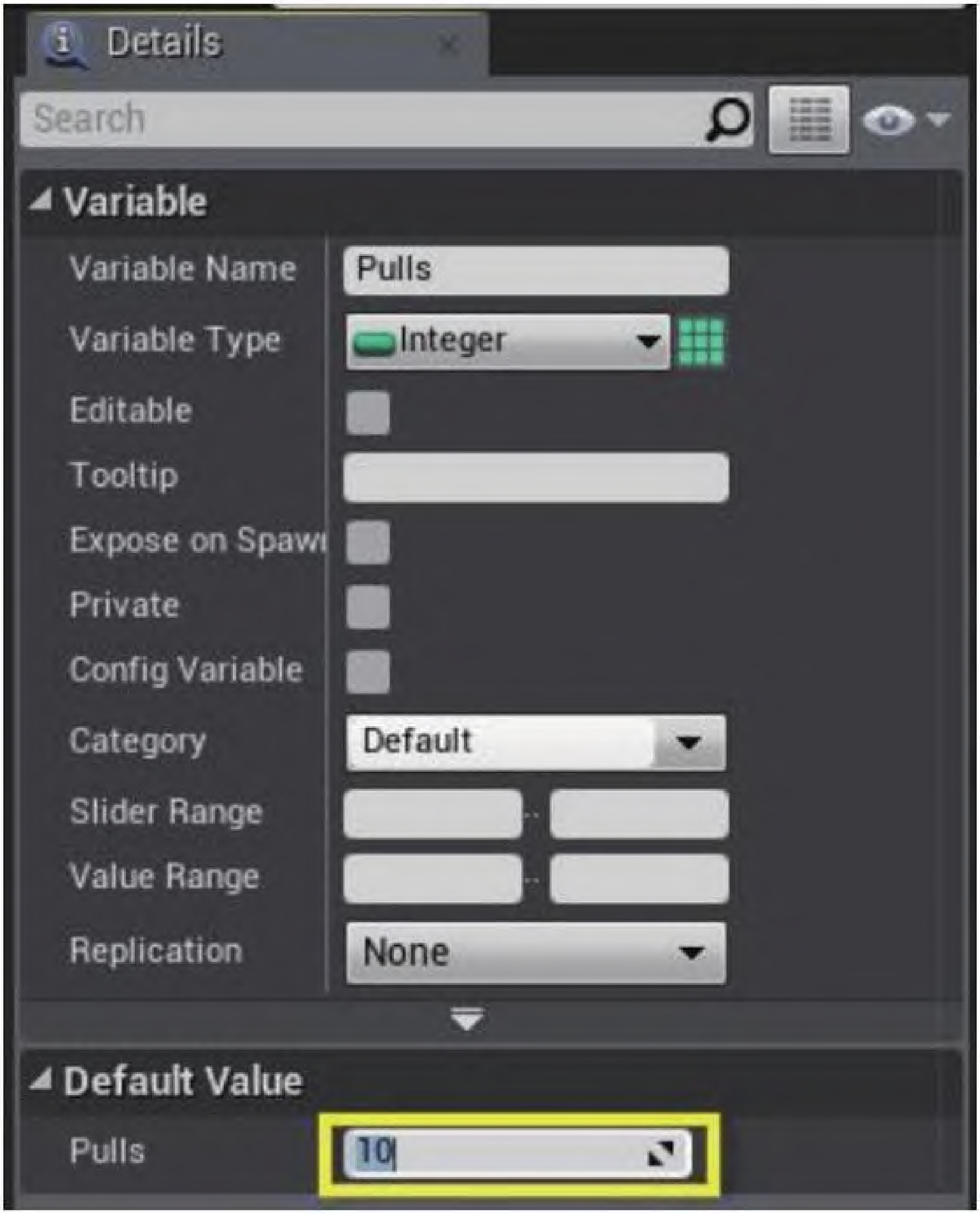



图4-36

**Step 12**　设置Shots的默认值为3，设置Score为0。

#### 4.2.3.2　武器发射设置

在这里，我们要对玩家能开枪的次数进行小小的变更。

**Step 1**　搜索InputAction Fire事件并将其拖动出来，然后添加Do N节点，如图4-37所示。

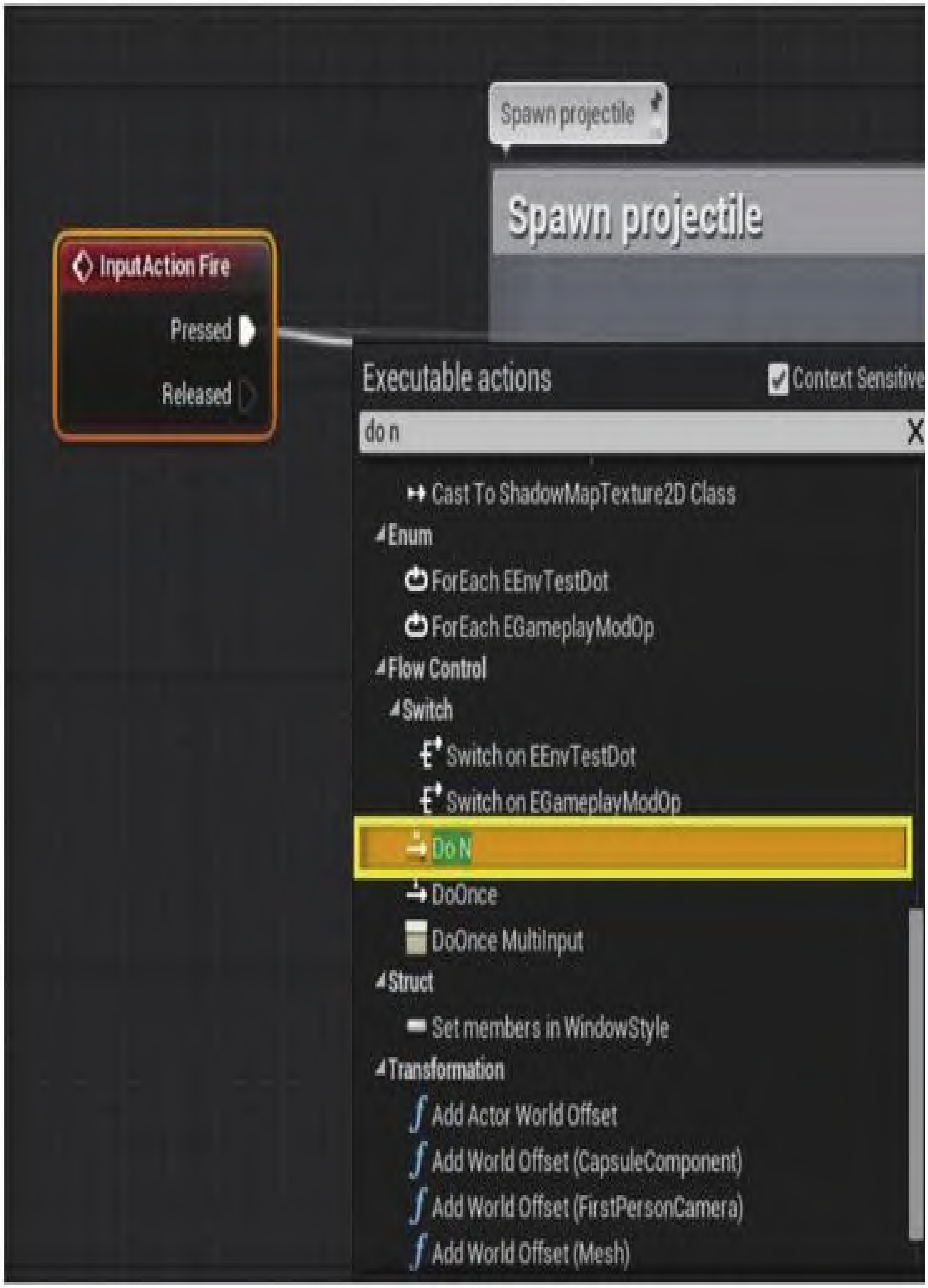

Spawn projectile
Spawn projectile
InputAction Fire
Pressed
Released
Executable actions
Context Sensitive
do n
Cast To ShadowMapTexture2D Class
Enum
ForEach EEnvTestDot
ForEach EGameplayModOp
Flow Control
Switch
Switch on EEnvTestDot
Switch on EGameplayModOp
Do N
DoOnce
DoOnce MultiInput
Struct
Set members in WindowStyle
Transformation
Add Actor World Offset
Add World Offset (CapsuleComponent)
Add World Offset (FirstPersonCamera)
Add World Offset (Mesh)

图4-37

这个节点会对脚本执行指定次数（从而让我们可以控制玩家发射的次数）。

**Step 2**　我们把N设置为3，然后在按住Alt键的情况下，把Shots变量从MyBlueprint窗口中拖入，然后重新连接节点，如图4-38所示。

在连接Shots节点后，把Set连接回Montage Play（播放动画剪辑）节点。

**Step 3**　在按住Ctrl键不放的情况下，再次拉入Shots变量，并将其连接到Int-Int节点（设置为1），然后按照显示内容进行连接，如图4-39所示。

图4-38

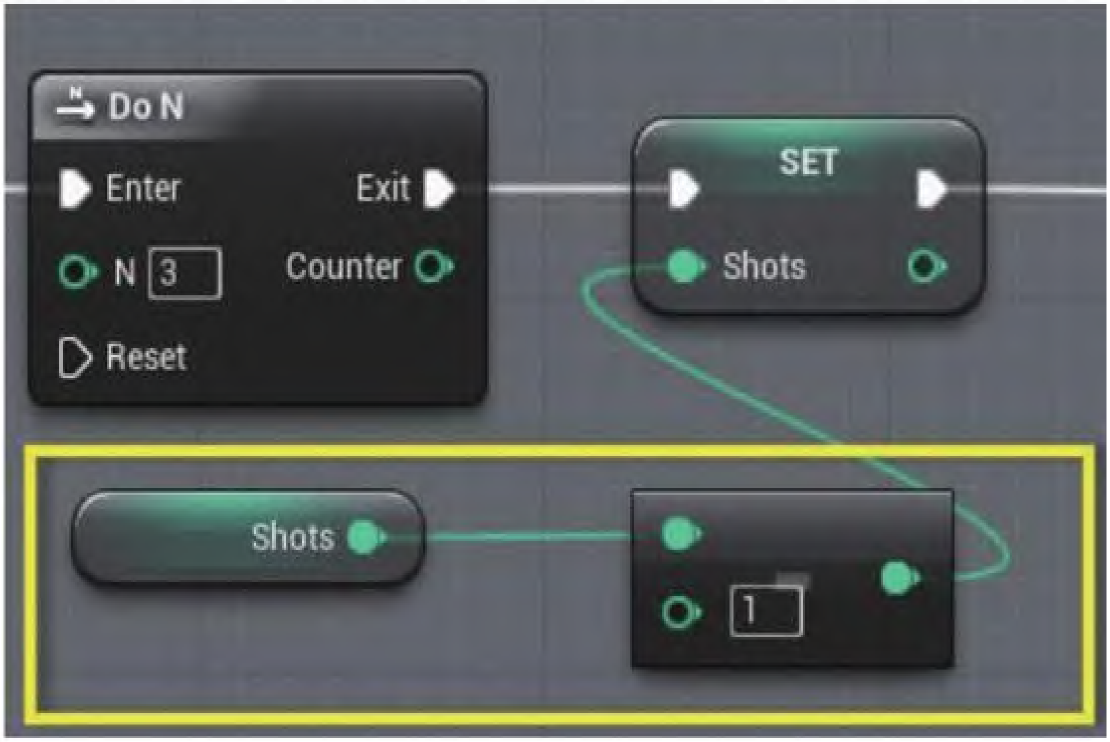



图4-39

在这里，我们把射击次数设置为与本身均等，每次按下Fire时都为-1。

**Step 4** 右击Shots旁的图表，然后添加名称为ResetShots（重设射击次数）的Custom Event（自定义事件），如图4-40所示。

图4-40

**Step 5**　按住Alt键，拉入Shots，并将其设置为3，然后先连接Custom Event（自定义事件），再连接到DoN节点的Reset（重设），如图4-41所示。

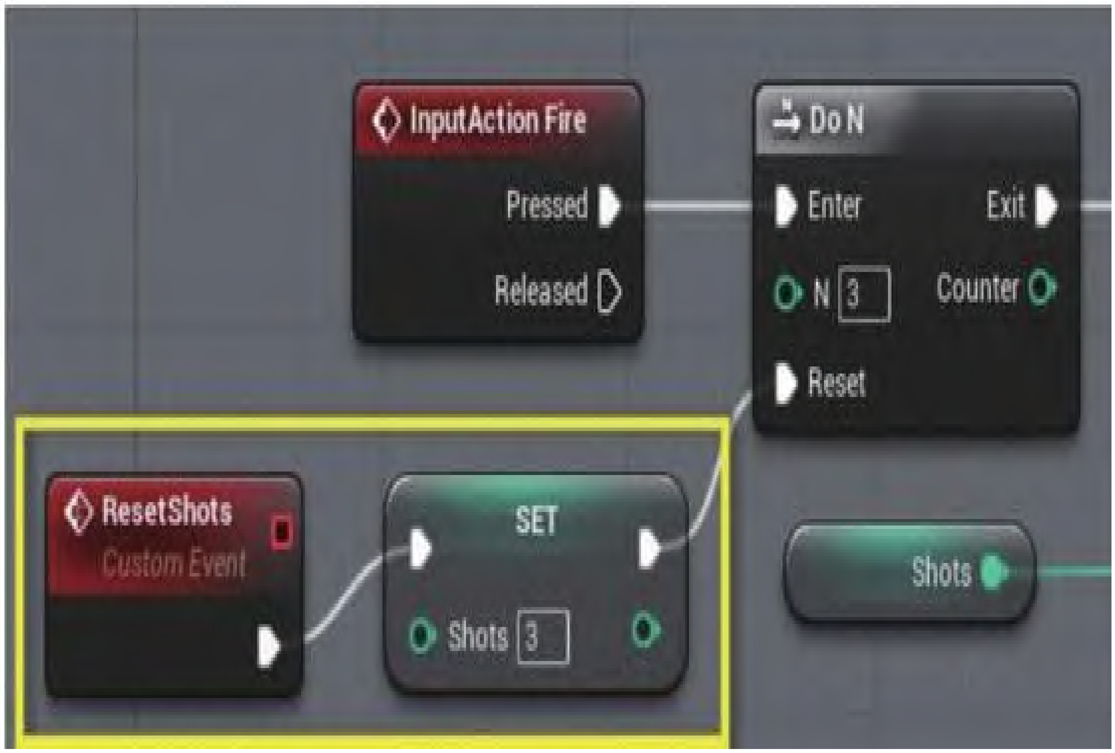



图4-41

当调用这个自定义事件时，它会把射击次数设置回3，然后重设DoN节点，这样就可以再次执行它。

## 4.2.4　生成可射击的岩石

在这个步骤中，我们会设置让玩家通过按钮来生成可射击岩石的逻辑（假设其他条件都已满足）。在关卡中，我们定义了一些生成点，并且为了获得这些位置之一，我们将会使用Event Dispatcher（事件调度器），它会在我们的Level Blueprint（关卡蓝图）中触发事件（我们会在下一步创建该事件），从而获得这些位置之一，并将其发送到我们的角色蓝图。

首先，让我们使角色可以通过按键来生成可射击的岩石。

**Step 1**　在FirstPersonCharacter蓝图中，右击图表中的空白空间，然后添加一个F关键事件，如图4-42所示。

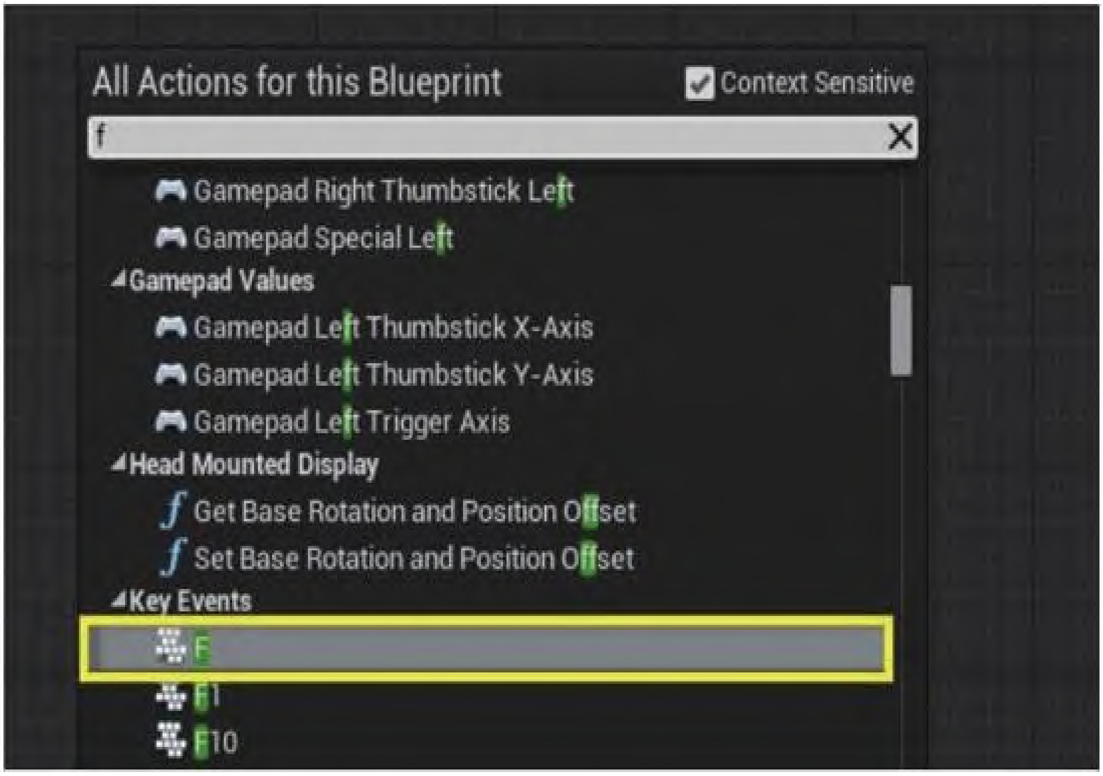



图4-42

**Step 2** 按住B键不放，然后在图表中左键单击，从而创建Branch（分支）节点。

**Step 3** 在isActive变量中按住Alt键并拖曳，然后勾选框，并按下方显示内容进行连接，如图4-43所示。

图4-43

**Step 4** 按住Ctrl键不放，并拖入isActive和Pulls变量。

**Step 5**　连接isActive到！=节点（已勾选），并连接Pulls到>节点（设置为0），如图4-44所示。

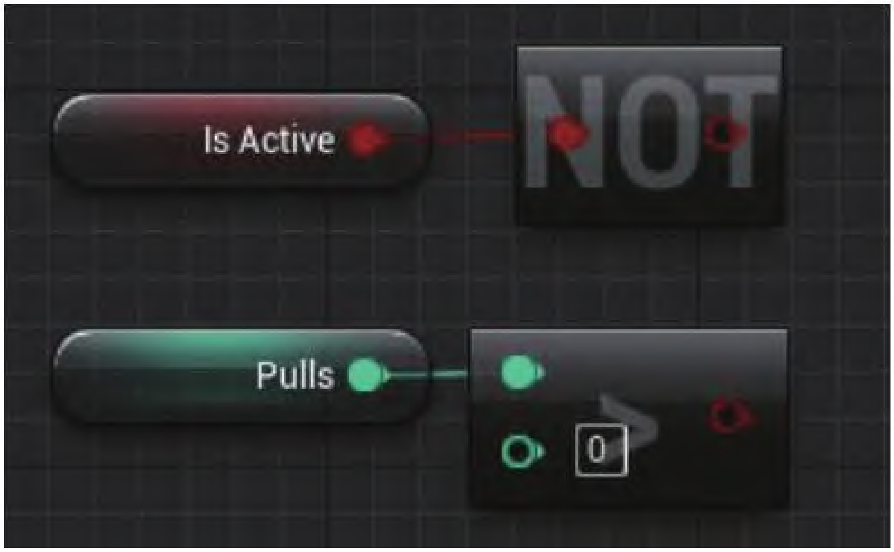



图4-44

**Step 6**　将两个输出都连接到And节点，并将其连接到Branch（分支）的Condition（条件）中，如图4-45所示。

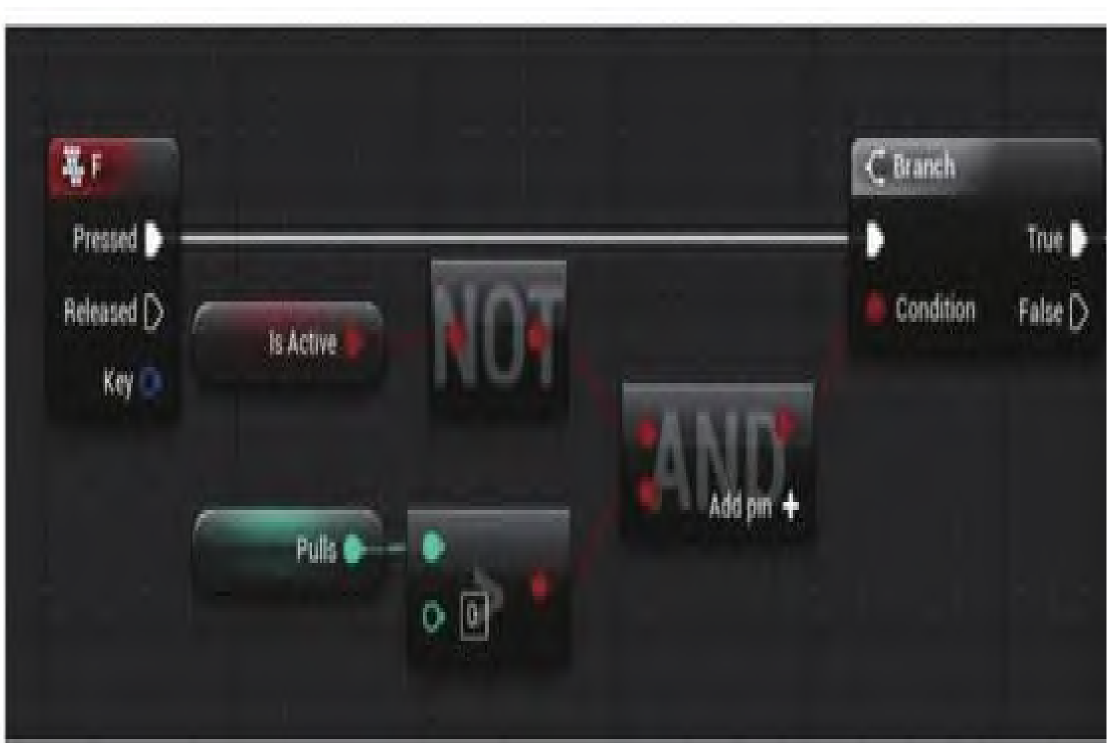



图4-45

在这里，我们设置了通过查看名称为isActive的变量是否为真，以及玩家的Pulls数量是否大于0来判断玩家是否可以生成岩石的逻辑。如果两者都为真，则我们会减少Branch（分支）节点运算结果为真的数量并生成岩石（当然，这部分的脚本仍需要进行设置，请阅下文）。

**Step 7**　按住Alt键并拖入Pulls变量，然后在Set IsActive节点连接后进行连接。

**Step 8**　在按住Ctrl键不放的情况下，再次拉入Pulls变量，并将其连接到Int-Int节点（设置为1）并连接到Set Pulls，如图4-46所示。

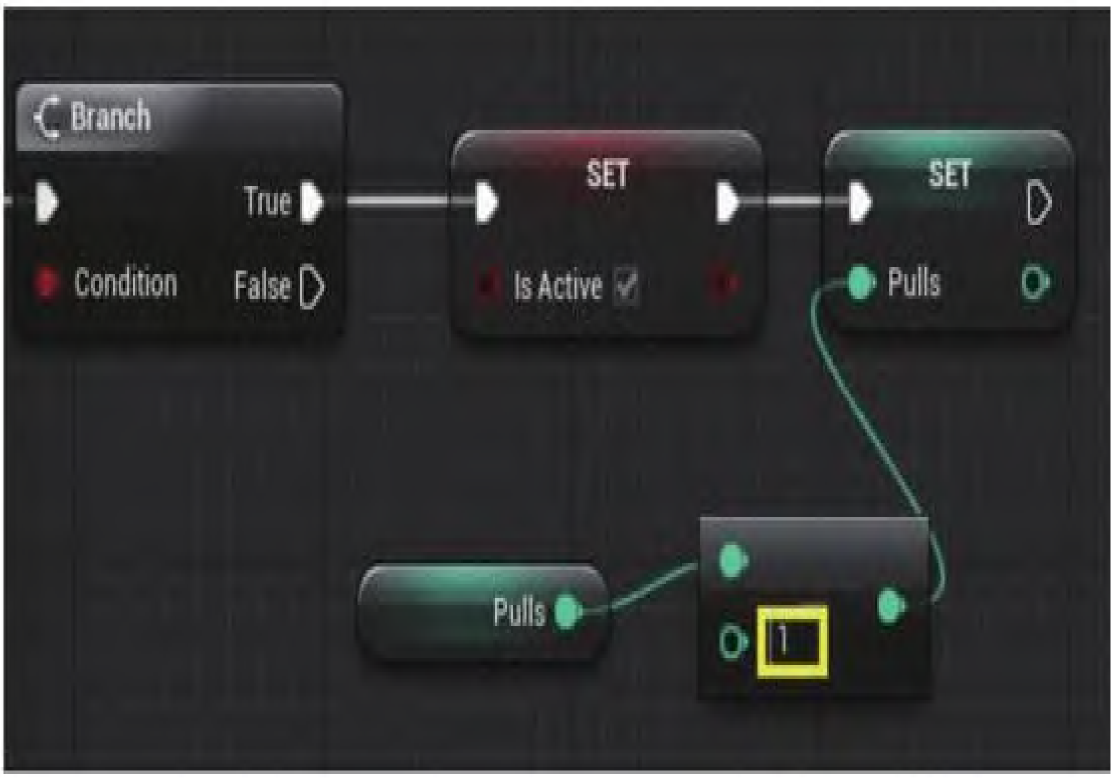



图4-46

**Step 9**　单击Compile（编译）按钮，然后关闭Set Pulls节点，添加ResetShots自定义事件，如图4-47所示。

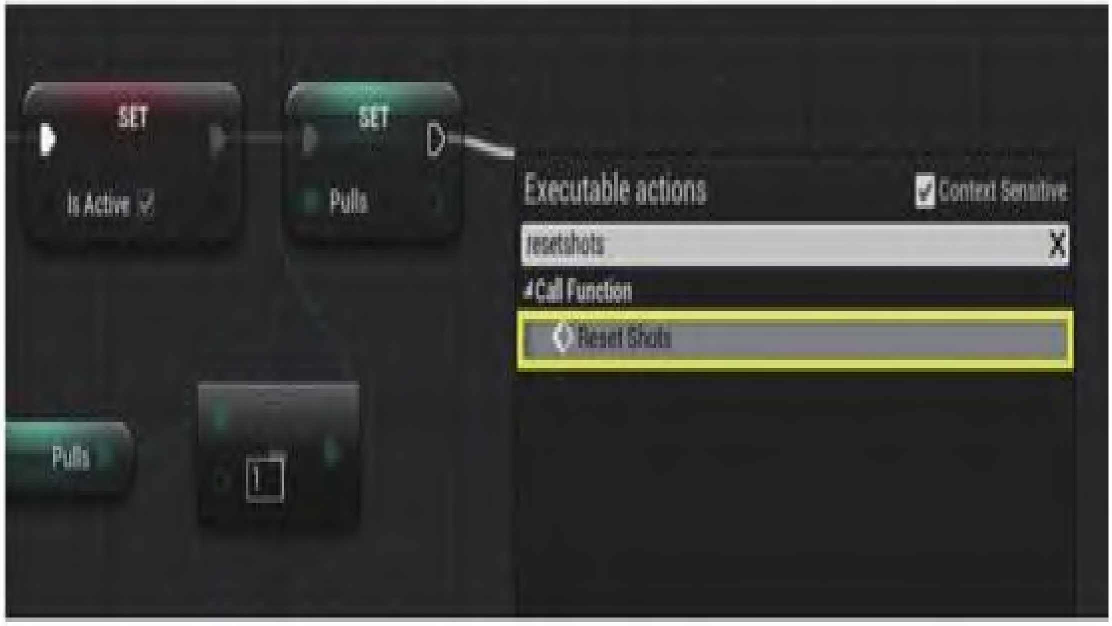



图4-47

你的设置应该看起来类似于以下内容，如图4-48所示。

图4-48

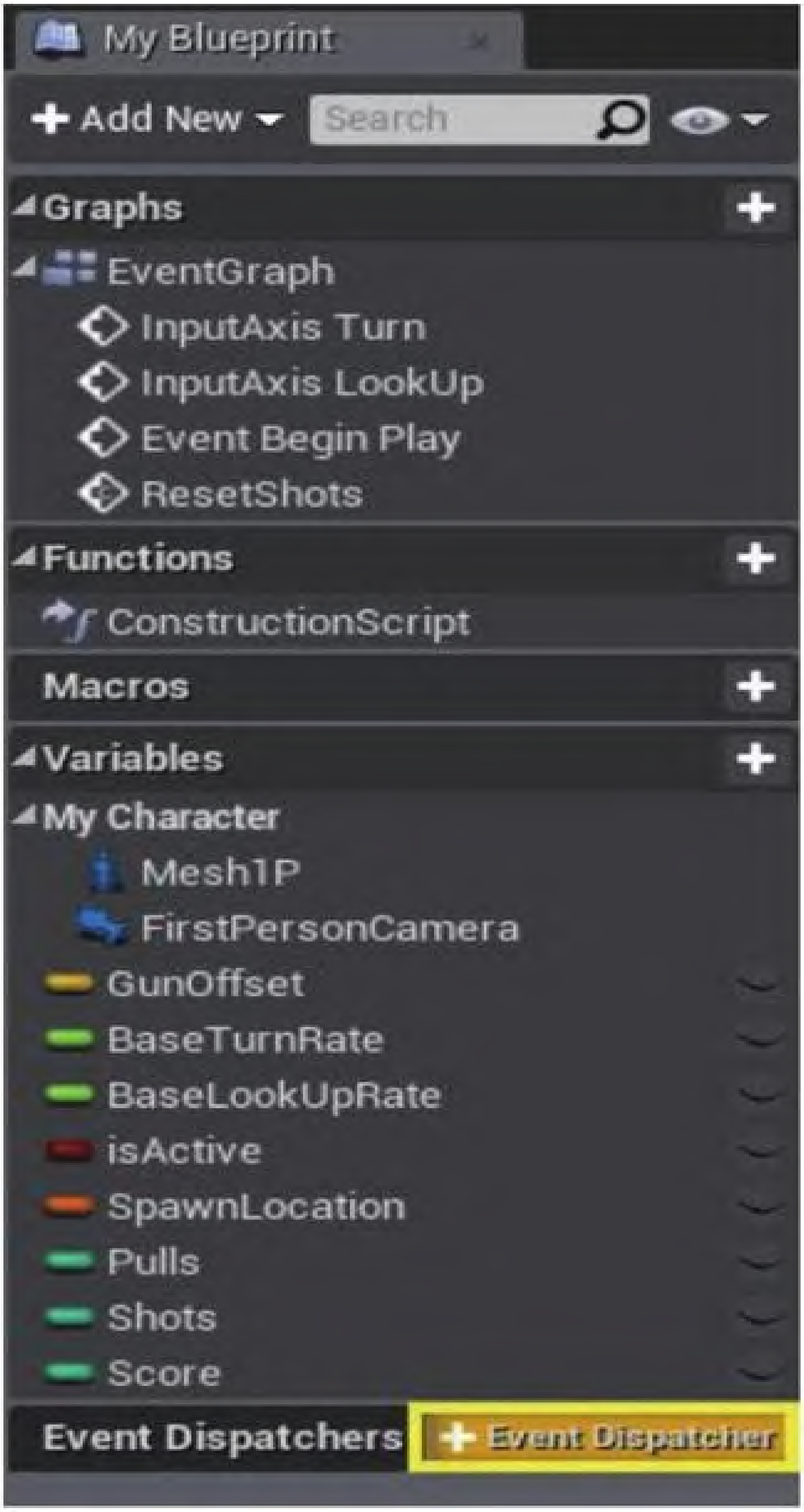
这样我们就设置完成了在玩家生成岩石时的逻辑，现在我们需要与关卡进行通信，从而获得生成位置。我们会使用Event Dispatcher（事件调度器）来执行此操作，它会在按下F键时进行调用。Level Blueprint（关卡蓝图）会监听事件调度器，而且当它被调用时，会触发其事件，从而获得生成点之一的位置，并将其传回角色蓝图。

**Step 10**　单击MyBlueprint窗口中的Add Event Dispatchers（添加事件调度器）按钮，如图4-49所示。

**Step 11**　单击新事件调度器并在选择后按下F2键，然后重命名为GetSpawnLocation，如图4-50所示。

**Step 12**　在Reset Shots（重设射击次数）函数后的图表中，添加Call GetSpawnLocation（调用GetSpawnLocation）节点，如图4-51所示。

此时将会添加事件调度器，这是由节点右上角处的信封图标来表示的，如图4-52所示。



My Blueprint
Add New
Search
Graphs
EventGraph
InputAxis Turn
InputAxis LookUp
Event Begin Play
ResetShots
Functions
ConstructionScript
Macros
Variables
My Character
Mesh1P
FirstPersonCamera
GunOffset
BaseTurnRate
BaseLookUpRate
isActive
SpawnLocation
Pulls
Shots
Score
Event Dispatchers
Event Dispatcher


图4-49

Step 13　在GetSpawnLocation节点下，添加Spawn Actor from Class（从类中生成Actor）节点，如图4-53所示。

图4-53

**Step 14** 在Spawn Actor（生成Actor）节点上，设置Class（类）为BP_Rock。

**Step 15** 另外在Spawn Actor节点上，拖放SpawnLocation变量到SpawnTransform引脚，如图4-54所示。

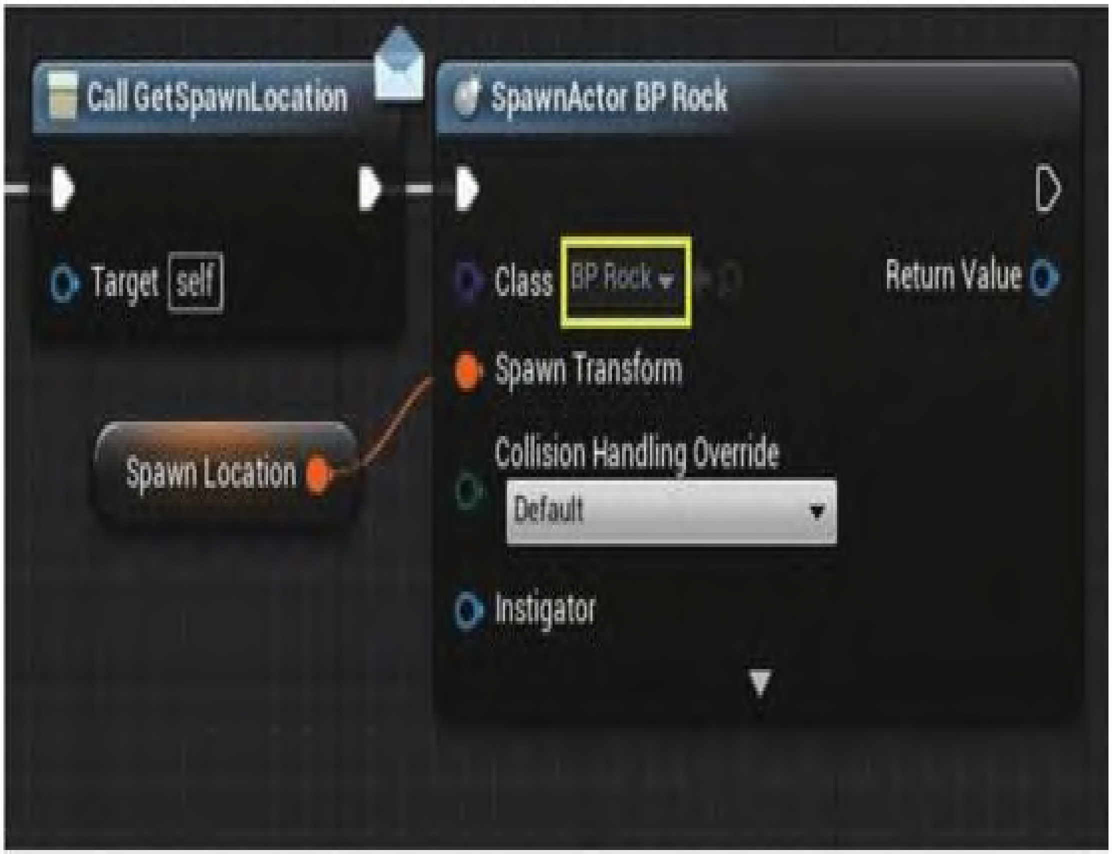



图4-54

现在我们的角色蓝图已经设置完成，可以生成可射击的岩石并获得生成岩石的Transform（变换）位置。

在下一步中，我们会使用Level Blueprint（关卡蓝图）来获得生成点之一的位置，并在调用角色蓝图的事件调度器时传入该信息到角色蓝图（这样我们就可以在角色蓝图和关卡蓝图间传递信息了）。

## 4.2.5　从关卡传递信息到角色

**Step 1**　在主编辑器视图的工具栏中，单击Blueprints（蓝图）按钮并选择Open Level Blueprint（打开关卡蓝图），如图4-55所示。

**Step 2**　在图表中，右击并添加Event Begin Play（开始播放事件）节点以及Get Player Character（获取玩家角色）节点，如图4-56所示。

Settings
Blueprints
Matinee
Build
Play
Launch
Blueprint Class
New Empty Blueprint Class...
Convert Selected Components to Blueprint Class...
Convert Selected Actor to Blueprint Class...
Open Blueprint Class...
Level Blueprints
Open Level Blueprint
Project Settings
GameMode: Edit FirstPersonGameMode
World Override
GameMode: Not overridden!
0.25

图4-55

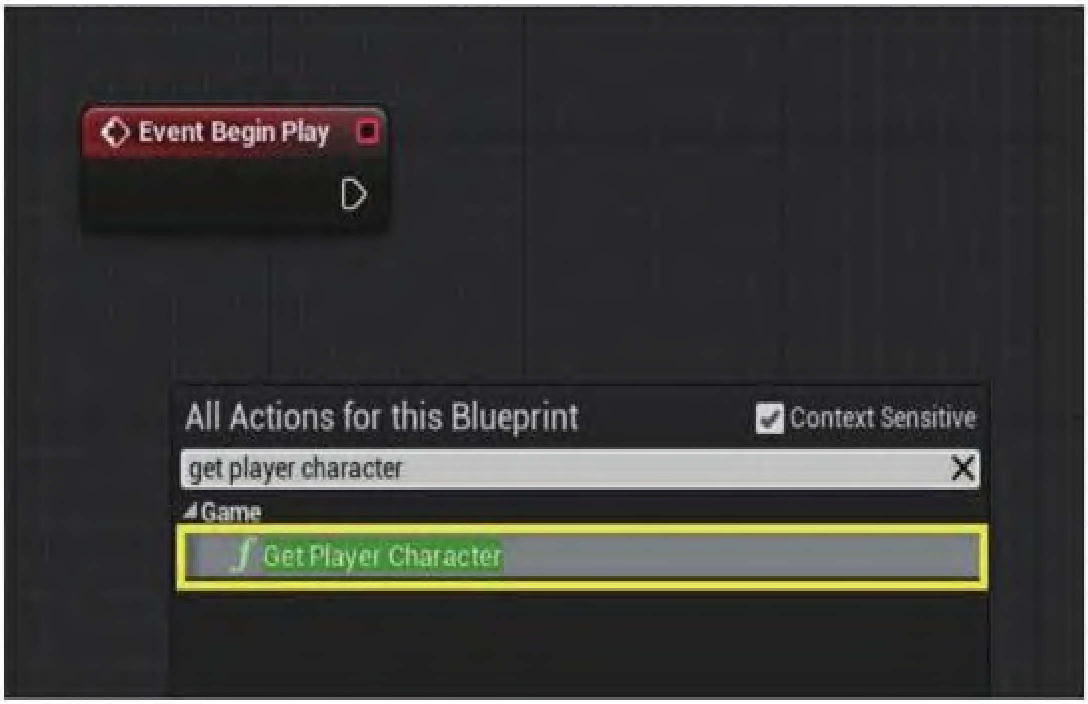



图4-56

**Step 3** 从Get Player Character（获取玩家角色）节点的Return Value（返回值）引脚进行拖动，并添加一个Cast To FirstPersonCharacter（投射到第一人称角色）节点，如图4-57所示。

图4-57

在此我们使用Cast（投射）节点来首先获取玩家角色，然后获取由该玩家使用的角色蓝图（本例中被称为FirstPersonCharacter，并且我们使用它来对可玩角色设置其他脚本）。通过使用投射节点，我们得以访问玩家角色的蓝图以及变量、事件和其中的函数。

**Step 4**　按照下面的内容来连接节点，然后在Cast To节点的Return Value（返回值）处，选择Promote to variable（添加到变量中），如图4-58所示。

图4-58

我们会创建一个变量，从而保存对玩家正在使用的FirstPersonCharacter蓝图的引用，这样我们就不必每次使用投射来获取结果。通过在Event Begin Play（开始播放事件）执行此操作，它将会在稍后为我们节省时间。

**Step 5**　选择添加的变量，然后在详细信息面板中，将其重命名为MyCharacter，如图4-59所示。

**Step 6**　在MyCharacter节点的Return Value（返回值）处，添加Assign Get Spawn Location（分配获取生成位置）节点，如图4-60所示。

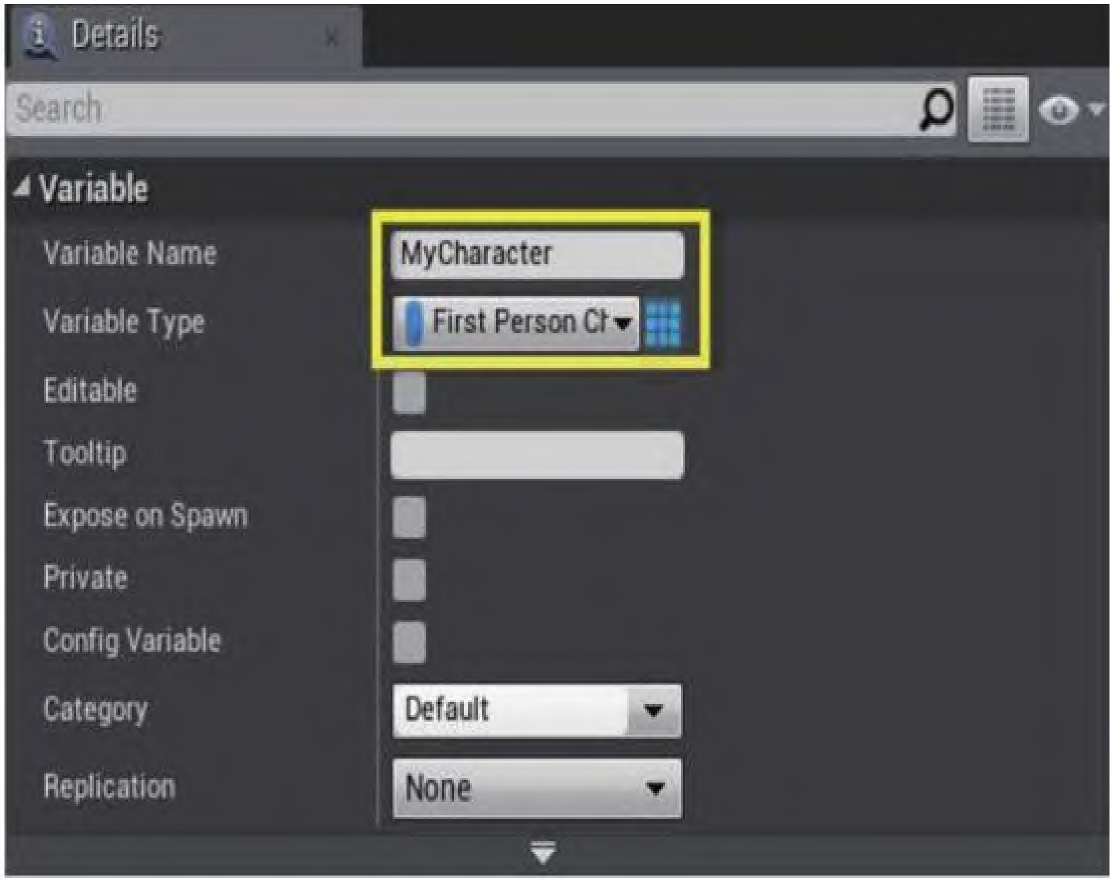



图4-59

图4-60

这就是我们创建的Event Dispatcher（事件调度器）。当你选择它时，将添加两个额外的节点。

如果你在列表中没有看到这个节点，请确认你单击了FirstPersonCharacter蓝图中的Compile（编译）按钮并再次尝试。

**Step 7** 请确认你连接了Bind Event（绑定事件）的新增加的节点，如图4-61所示。

现在，当我们在角色蓝图中调用GetSpawnLocation时，也会同时调用这个新的自定义事件。

**Step 8** 按住Ctrl键，从MyBlueprint窗口中拖入MyCharacter变量，然后添加Set Spawn Location（设置生成位置）节点，如图4-62所示。

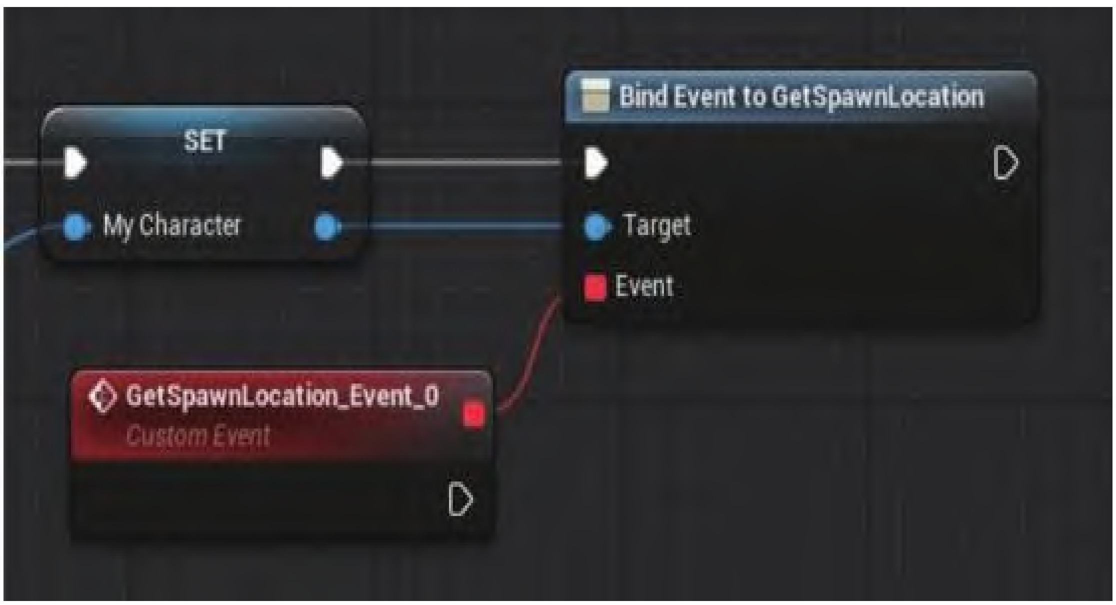



图4-61

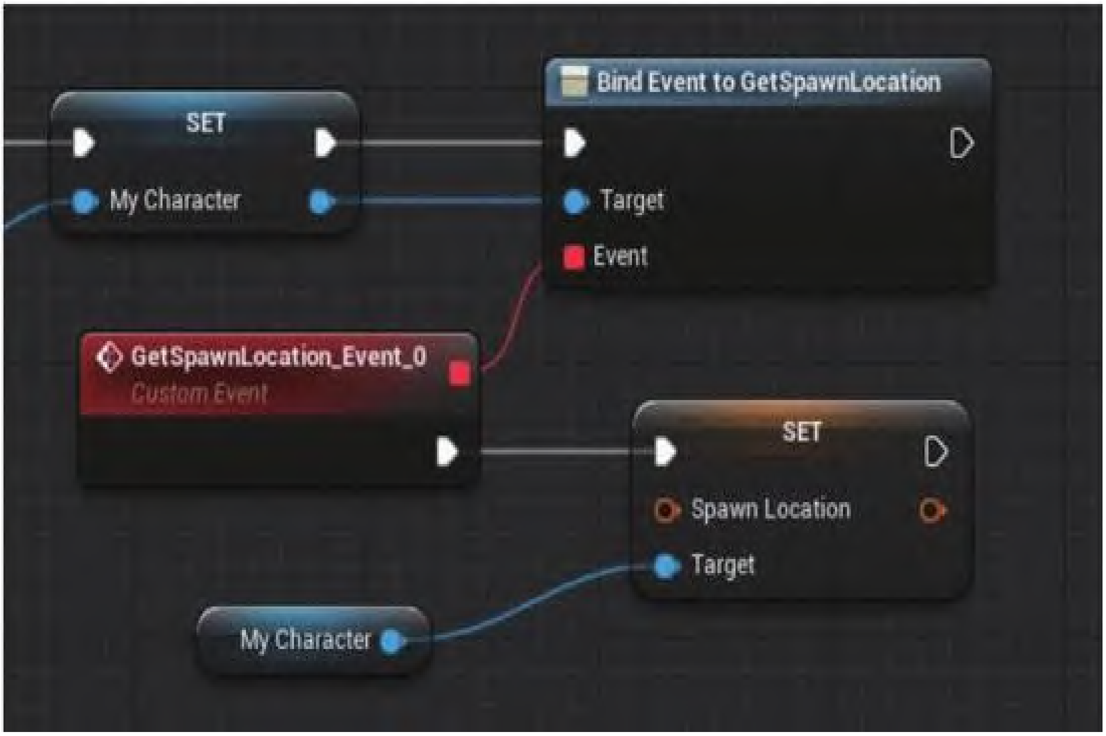



图4-62

**Step 9** 返回主编辑器窗口，并在World Outliner（世界大纲视图）中，搜索并选择每个Target Points（目标点），如图4-63所示。

**Step 10** 返回Level Blueprint（关卡蓝图）并在图表中右击并选择Create References to3 selected Actors（创建到3个选择的Actor的引用），如图4-64所示。

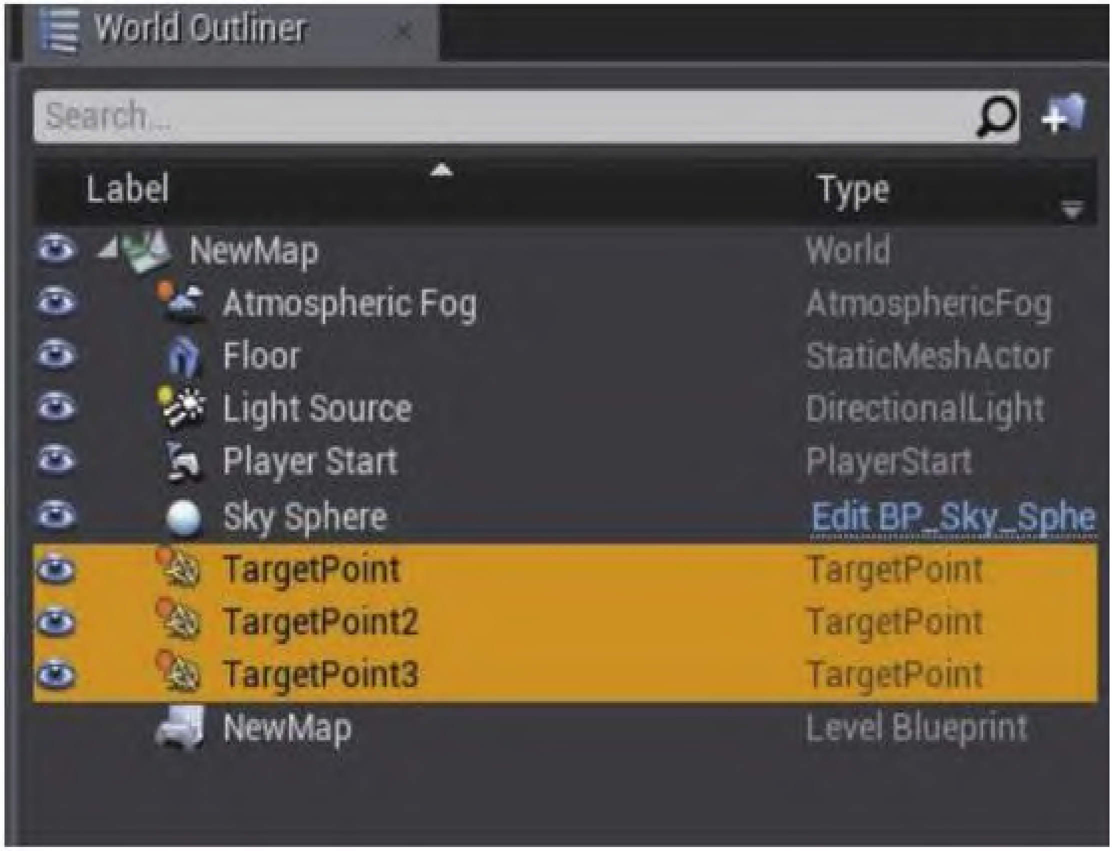

World Outliner
Search...
Label
Type
NewMap
World
Atmospheric Fog
AtmosphericFog
Floor
StaticMeshActor
Light Source
DirectionalLight
Player Start
PlayerStart
Sky Sphere
Edit BP_Sky_Sphe
TargetPoint
TargetPoint
TargetPoint2
TargetPoint
TargetPoint3
TargetPoint
NewMap
Level Blueprint

图4-63

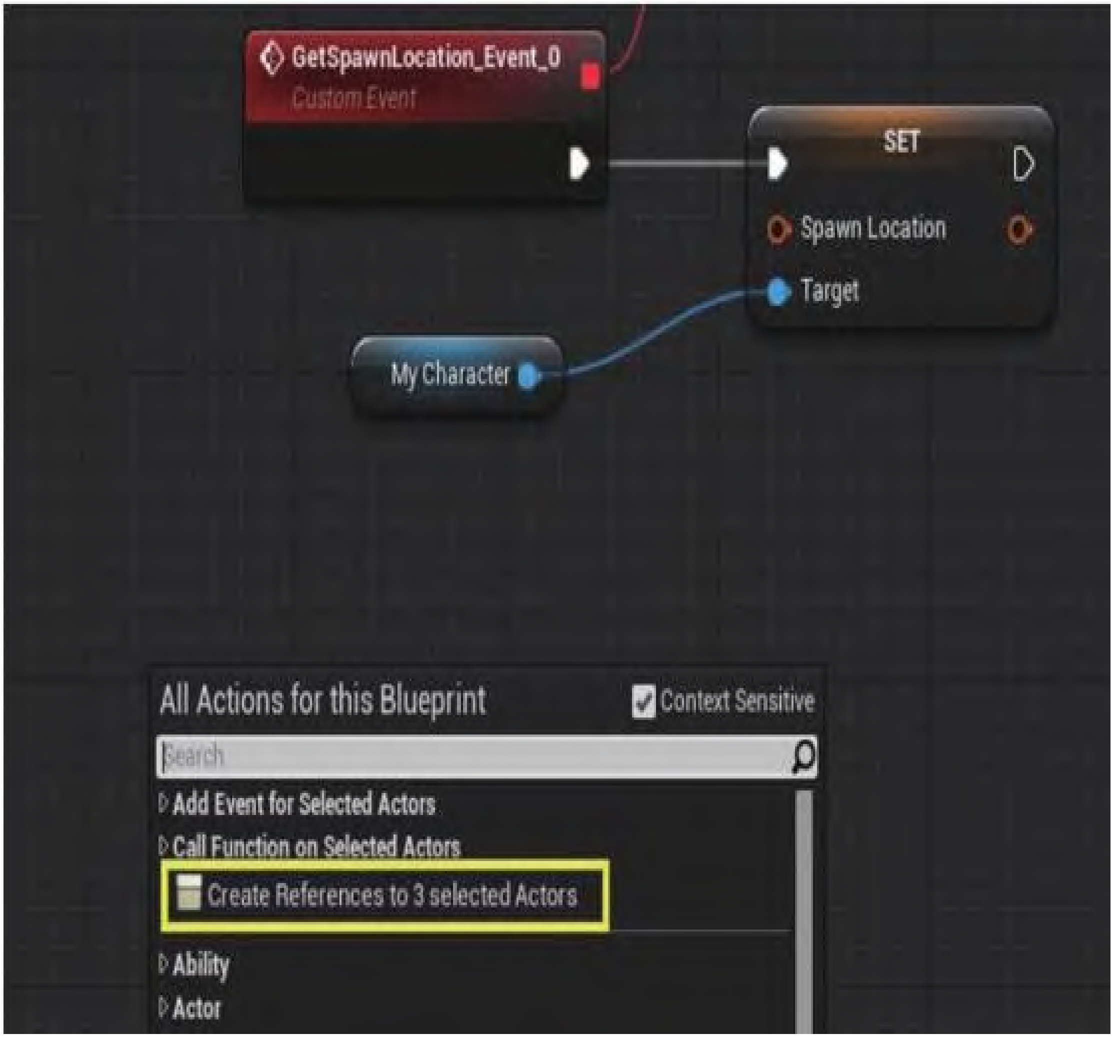



图4-64

**Step 11**　在图表中右击并添加Make Array（生成数组）节点，然后将其与每个Target Points（目标点）连接，如图4-65所示。

你可以通过单击Add Pin（添加引脚）按钮来添加更多引脚。这个数组会存储我们可以访问的目标点集合。

**Step 12**　在Array（数组）引脚，添加Get（获取）节点。

**Step 13**　在Get节点，添加Random Integer in Range（范围内的随机整数）节点，Min设置为0，Max设置为2，如图4-66所示。

图4-65

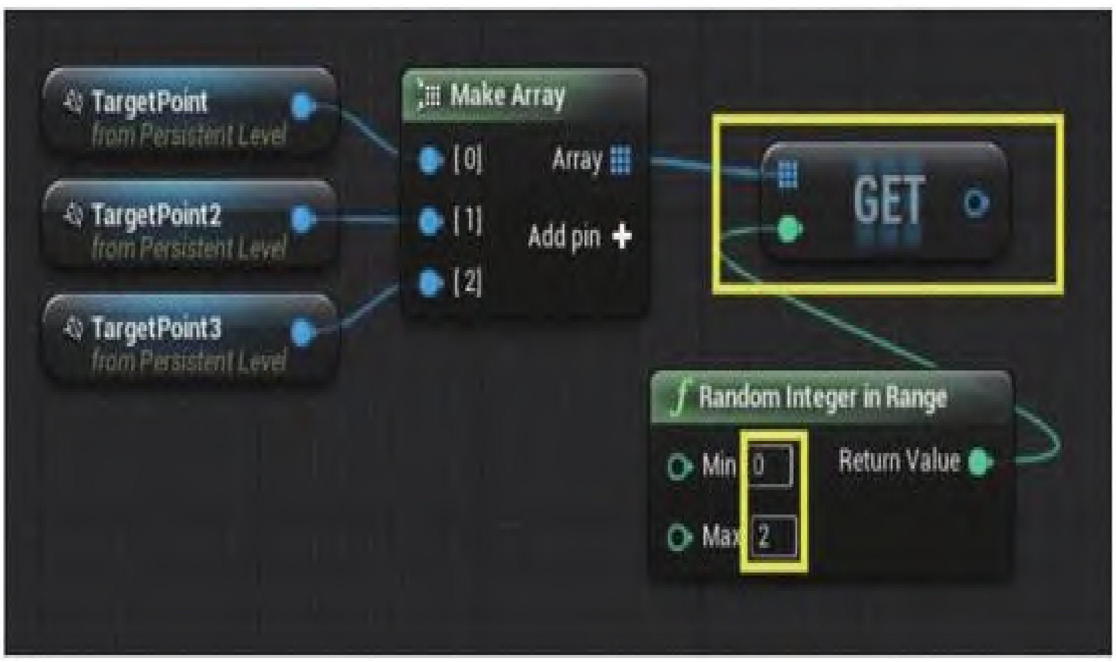



图4-66

在这里我们获得了可用作生成点的随机目标点。如果你有更多的生成位置，你可以将它们添加到数组中，并增大Range（范围）节点中的Max值从而包含它们。

**Step 14**　从Get节点，添加Get Actor Transform（获取Actor变换）节点，并将其连接到Set Spawn Location（设置生成位置）节点，如图4-67所示。

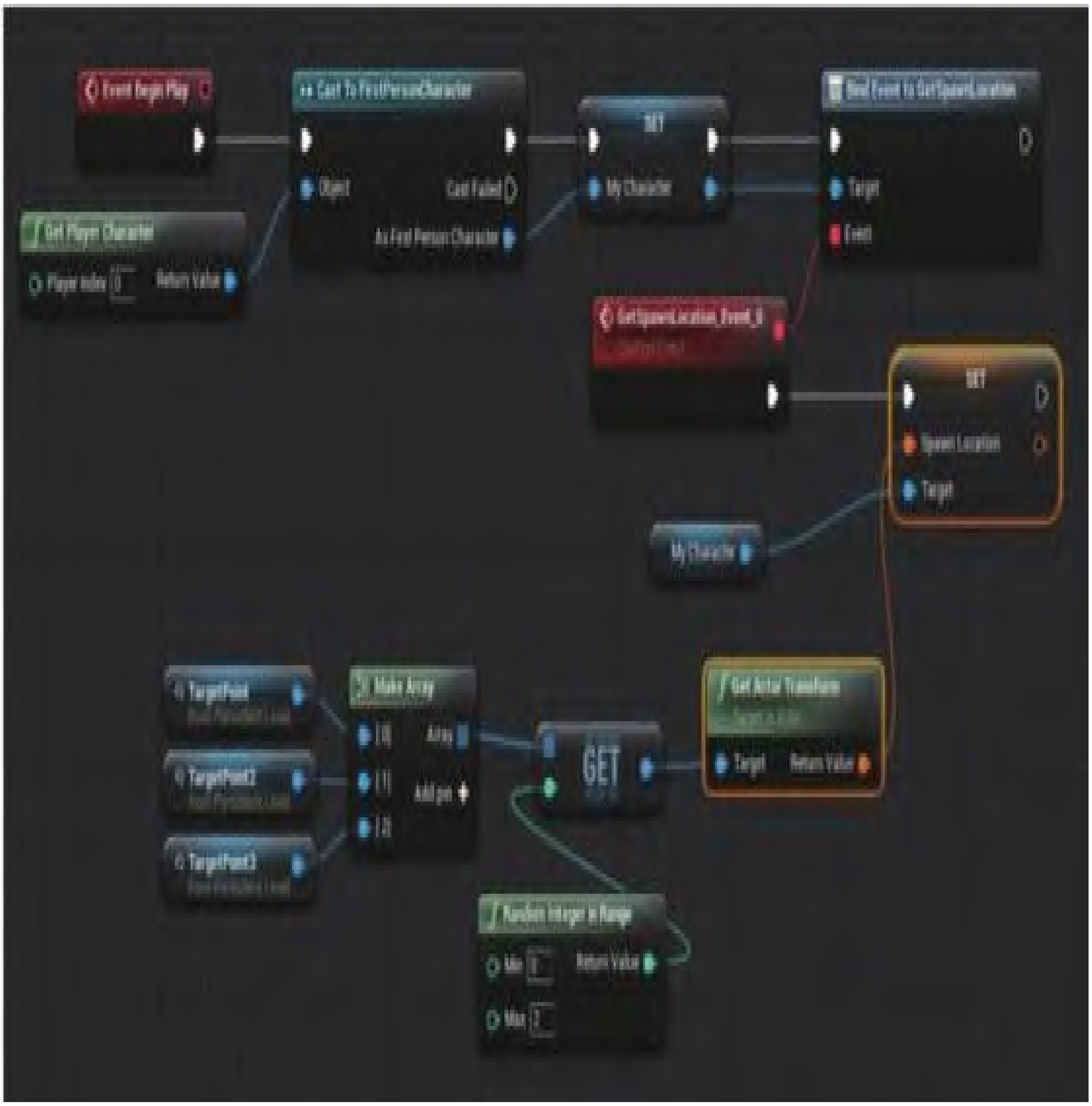



图4-67

你的完整脚本应该和上图类似，请单击以查看完整内容。

在这个页面里我们进行了许多操作！我们使用Cast节点来访问角色蓝图，然后绑定自定义事件到其中的事件调度器。这样我们可以在事件调度器被调用时，触发关卡蓝图中的自定义事件。我们随后获取随机生成位置之一，并将其放置在角色蓝图中作为变量生成位置（它用于我们生成岩石的位置）。

在下一步中，我们会让岩石变得可射击，并能够使用名称为OnHit的绑定事件进行销毁（也就是在岩石被射击时的事件）。

## 4.2.6　销毁岩石

在这个步骤中，我们会让岩石在被玩家发射的射弹击中时进行销毁。为了完成这项目标，我们会使用另一个Bind Event（绑定事件），但我们不会将其绑定到事件调度器。我们会转而使用一个名称为OnHit的碰撞事件，它会在岩石被射中时触发。

让我们开始射击吧！

**Step 1** 在FirstPersonCharacter蓝图中，在我们生成岩石的位置处搜索Spawn Actor（生成Actor）节点。

**Step 2** 在Spawn Actor（生成Actor）节点的Return Value（返回值）处拉根线，然后选择Assign On Actor Hit（在Actor碰撞时分配），如图4-68所示。

图4-68

**Step 3** 连接Spawn Actor（生成Actor）和Bind Event（绑定事件）。

**Step 4** 在Spawn Actor（生成Actor）节点的Return Value（返回值）处拉根线，然后选择Apply Radius Damage（Destructible）（应用半径伤害（可破坏物体）），如图4-69所示。

我们可以使用应用伤害，但还需要输入一些额外的参数。为了简单起见，我们会使用Radius Damage（半径损伤）。

**Step 5** 在Apply Radius Damage（应用半径损伤）节点，对下方显示的每个域我们都输入50000，然后连接Get Actor Location（获取Actor位

置）节点，如图4-70所示。

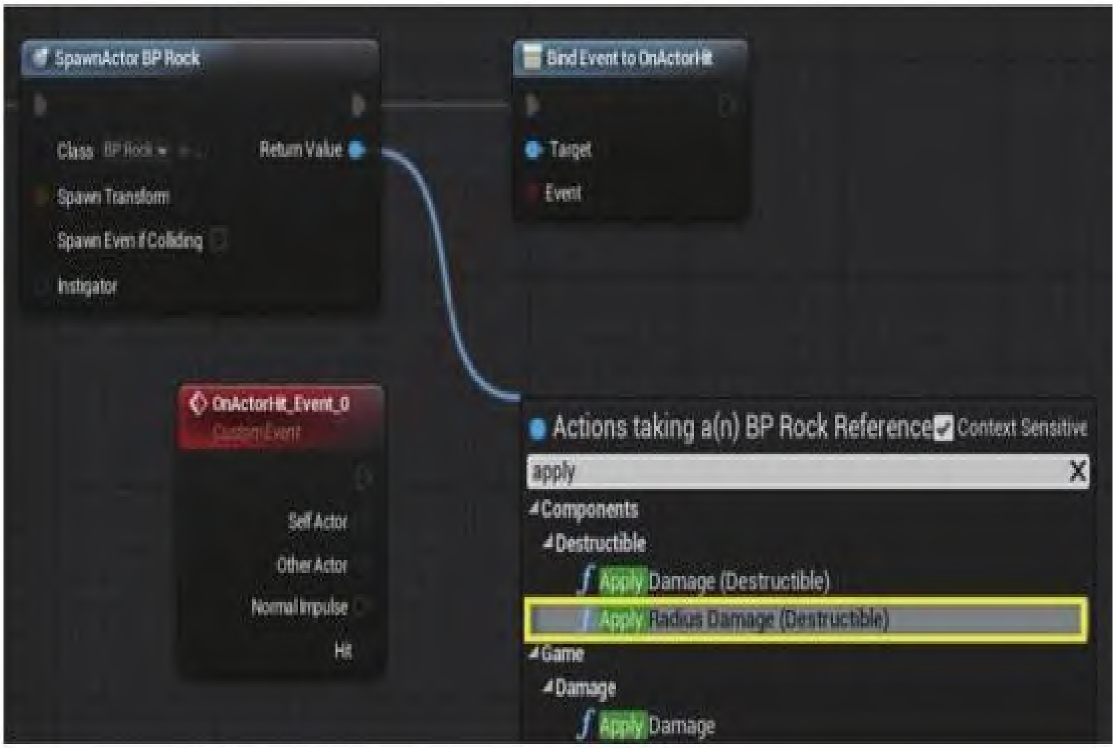



图4-69

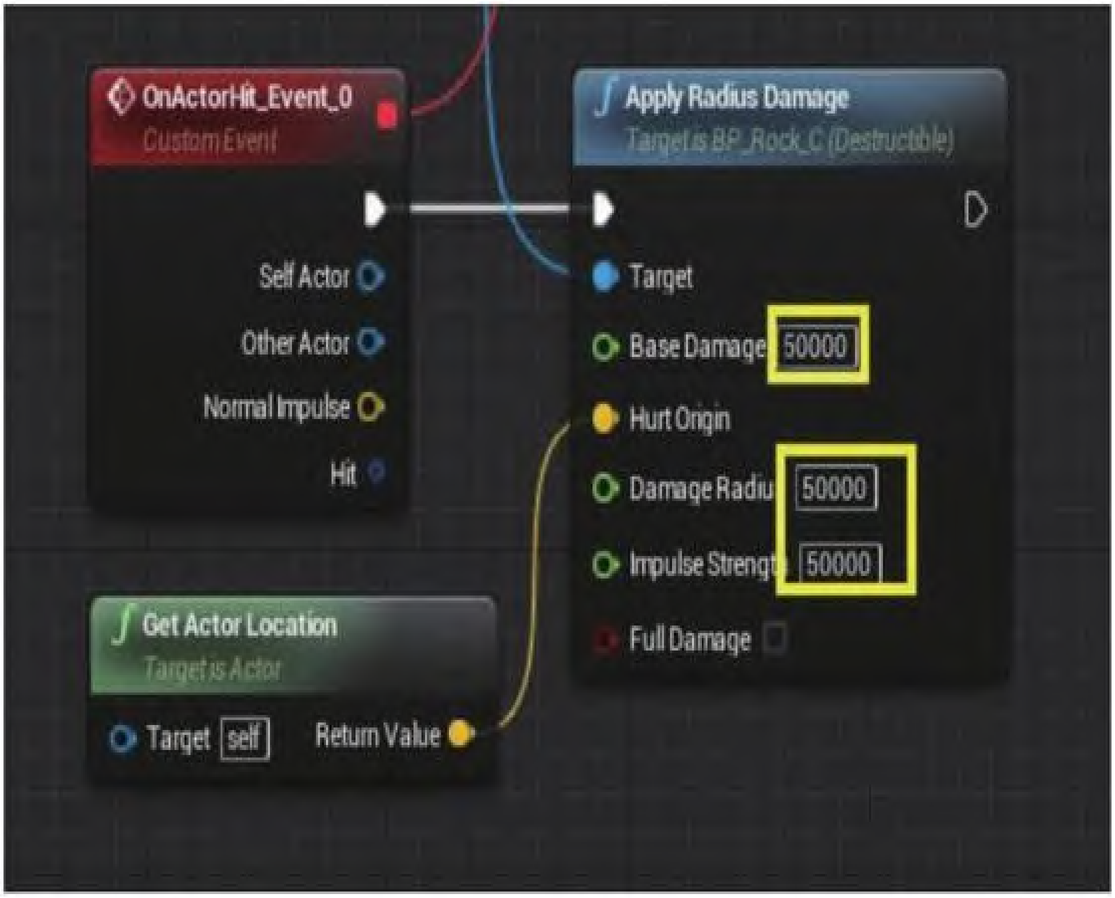



图4-70

这样我们就可以对岩石做出足够大的伤害，从而让岩石碎成一块一块。

**Step 6** 在Apply Radius Damage（应用半径伤害）节点之后，添加一个Set Score（设置分数）节点。

**Step 7** 在按住Ctrl键不放的情况下，再次拉入Score变量，并将其连接到Int-Int节点（设置为1）并连接节点，如图4-71所示。

这样我们就可以获得当前的分数，并在岩石每次被撞击时增加1值。

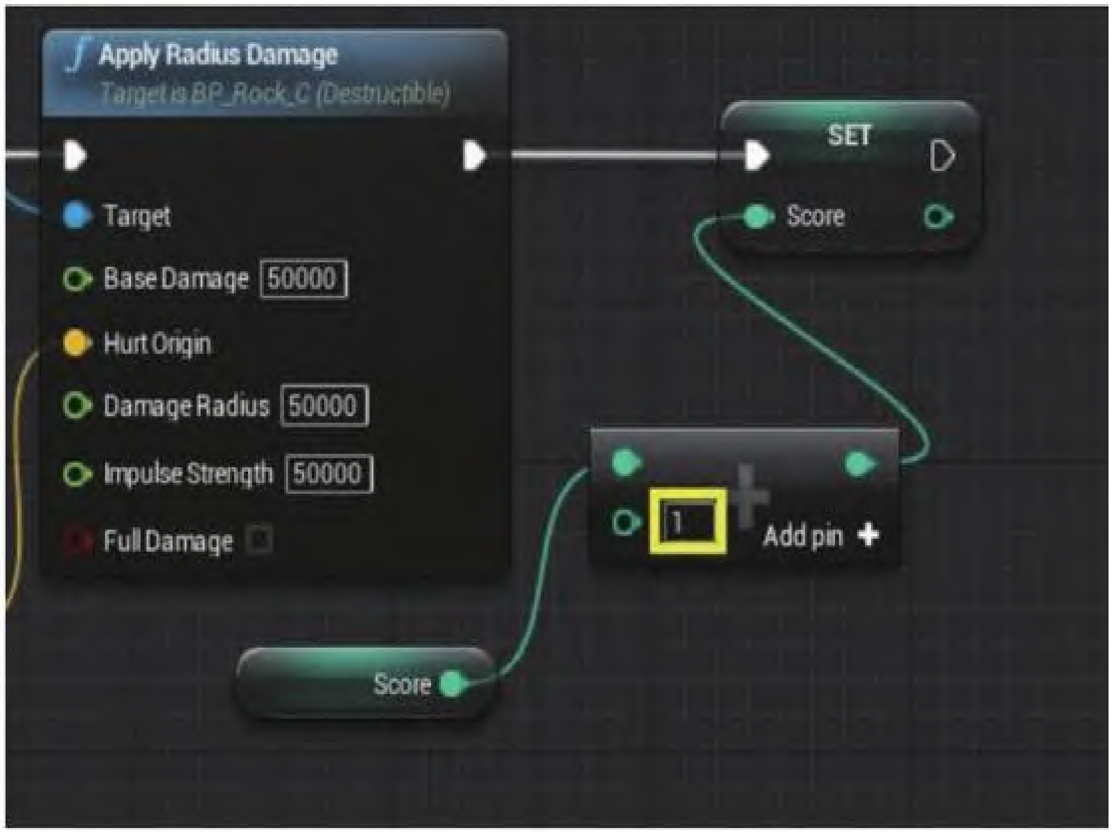



图4-71

**Step 8** 连接Bind Event（绑定事件）和Set Score（设置分数）节点到Delay（延迟）节点并设置其为3秒，如图4-72所示。

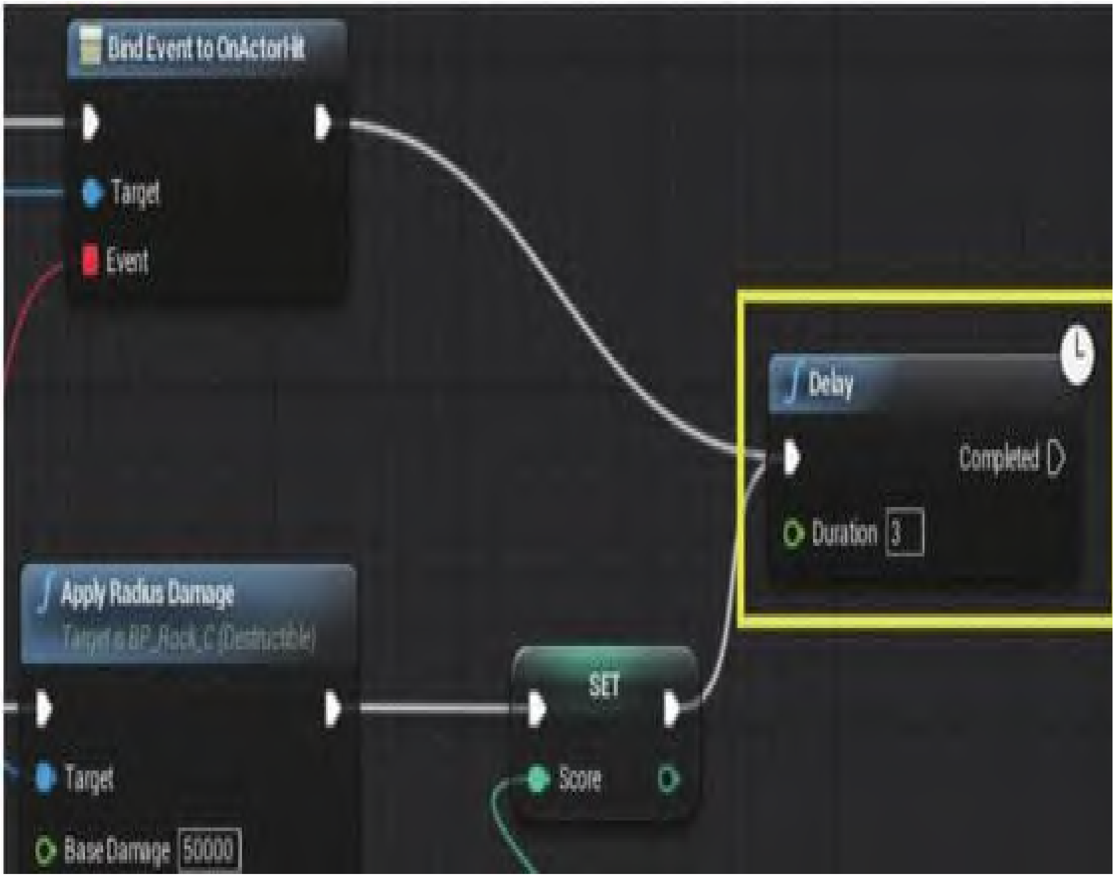



图4-72

在3秒后，当岩石生成后，无论其是否被击中，我们都会销毁它。

**Step 9** 给Delay（延迟）节点添加Destroy Actor（销毁Actor）节点，如图4-73所示。

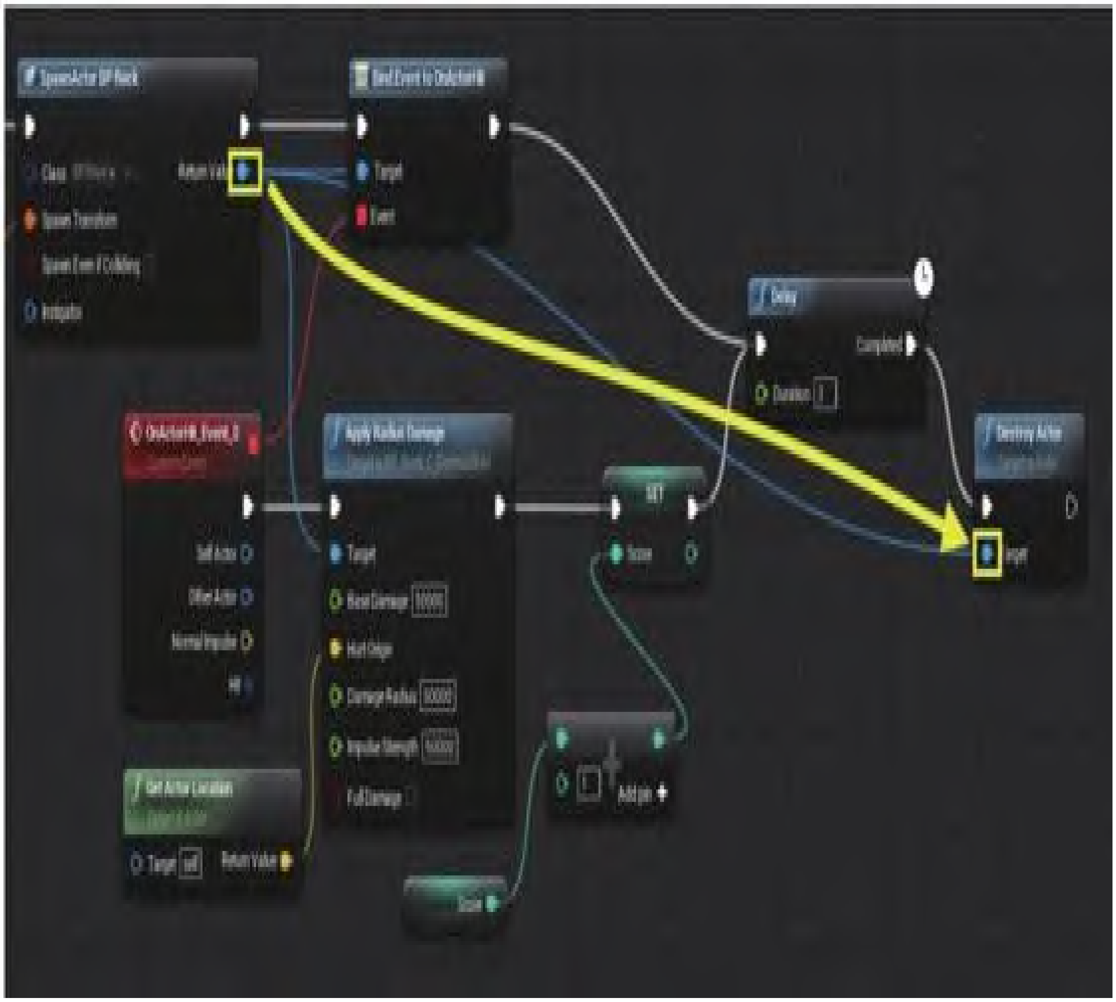



图4-73

在Spawn Actor（生成Actor）节点的Return Value（返回值）处拉根线，然后连接到Destroy Actor（销毁Actor）节点的Target（目标）位置。

**Step 10** 在Destroy Actor（销毁Actor）节点之后，添加Set is Active变量节点并保持其为未勾选的状态，如图4-74所示。

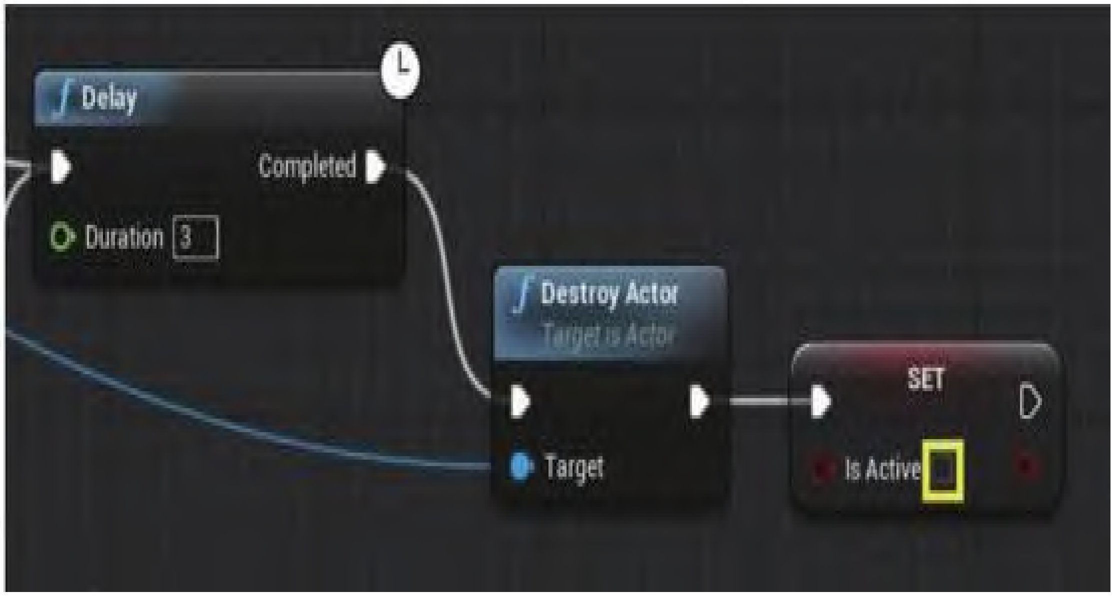



图4-74

也就是说，生成的岩石不再处于被激活状态，这样玩家可以生成另一块岩石（如果他们有足够的扳机扣动次数残留的话）

这样核心游戏就设置完成了，如果我们现在要在编辑器中进行播放，你可以看到，你可以显示生成、射击和销毁在关卡中随机生成点生成的岩石。在下一步中，我们将会与HUD进行通信，并基于来自角色蓝图的值来更新显示的值。

## 4.2.7 使用HUD通信

在这个步骤中，我们会通过与角色蓝图进行通信来获得Pulls、Shots和Score中存储的值，从而更新显示于HUD控件蓝图的值。为完成这个目标，我们使用Cast（投射）节点来访问玩家的角色蓝图，然后绑定HUD中的属性到角色蓝图中的值，这样它们会在这些值在角色蓝图中进行更新时自动获取更新。

游戏快完成了，最后让我们更新一下HUD。

**Step 1** 打开你创建的HUD控件蓝图。

**Step 2** 打开你的HUD控件蓝图中的Graph（图表）选项卡，如图4-75所示。

**Step 3** 在图表中右击并添加Event Construct（事件构建）节点，如图4-76所示。

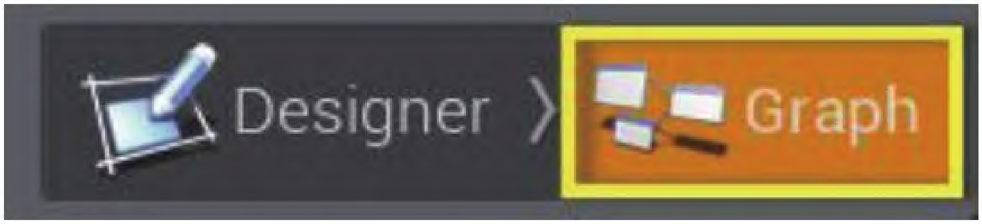



图4-75

图4-76

这个节点类似于Event Begin Play（开始播放事件）节点，并且会在控件蓝图被构建时进行调用。

**Step 4** 添加Get Player Character（获取玩家角色）节点和CastTo FirstPersonCharacter（投射到第一人称角色）节点，如图4-77所示。

图4-77

**Step 5**　在As First Person Character（作为第一人称角色）引脚，选择Promote to variable（添加为变量）并重命名新节点为MyCharacter，如图4-78所示。

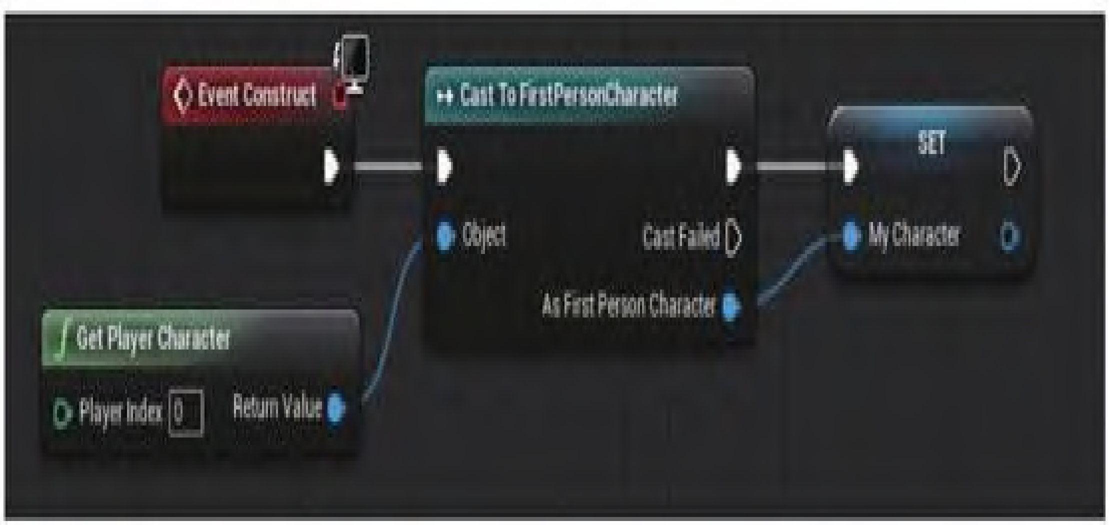



图4-78

你可以通过选择变量并按下F2键来从MyBlueprint窗口重命名变量。

**Step 6** 在Designer（设计）选项卡，单击Pulls的值，然后在详细信息面板旁的文本处，单击绑定选项，如图4-79所示。

图4-79

这将为绑定打开一个新的图表。

**Step 7** 按住Ctrl键，然后拖入MyCharacter变量，然后把Get Pulls连接到Return Node（返回节点）的Return Value（返回值），如图4-80所示。

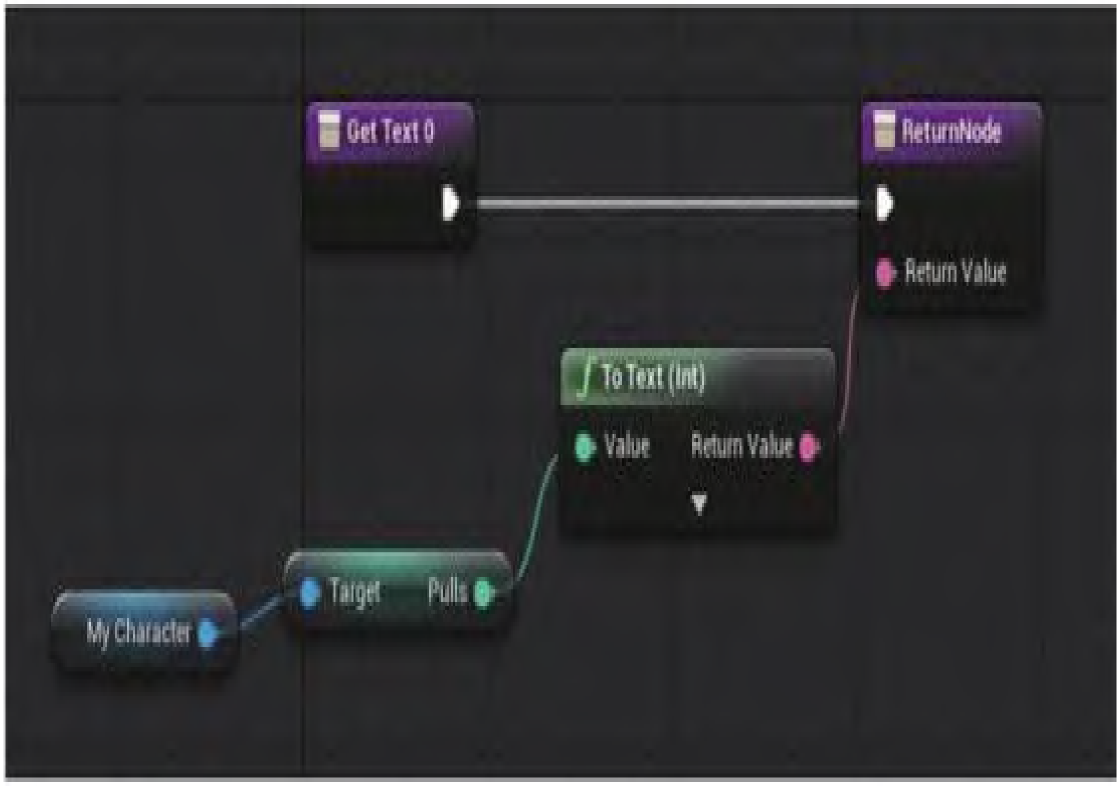



图4-80

To Text（Int）转换节点将会被自动创建。我们现在已经可以正确显示Pulls的值。

**Step 8**　在Designer（设计）选项卡，单击设置Shots的值，然后在详细信息面板，单击绑定选项并创建下方的绑定，如图4-81所示。

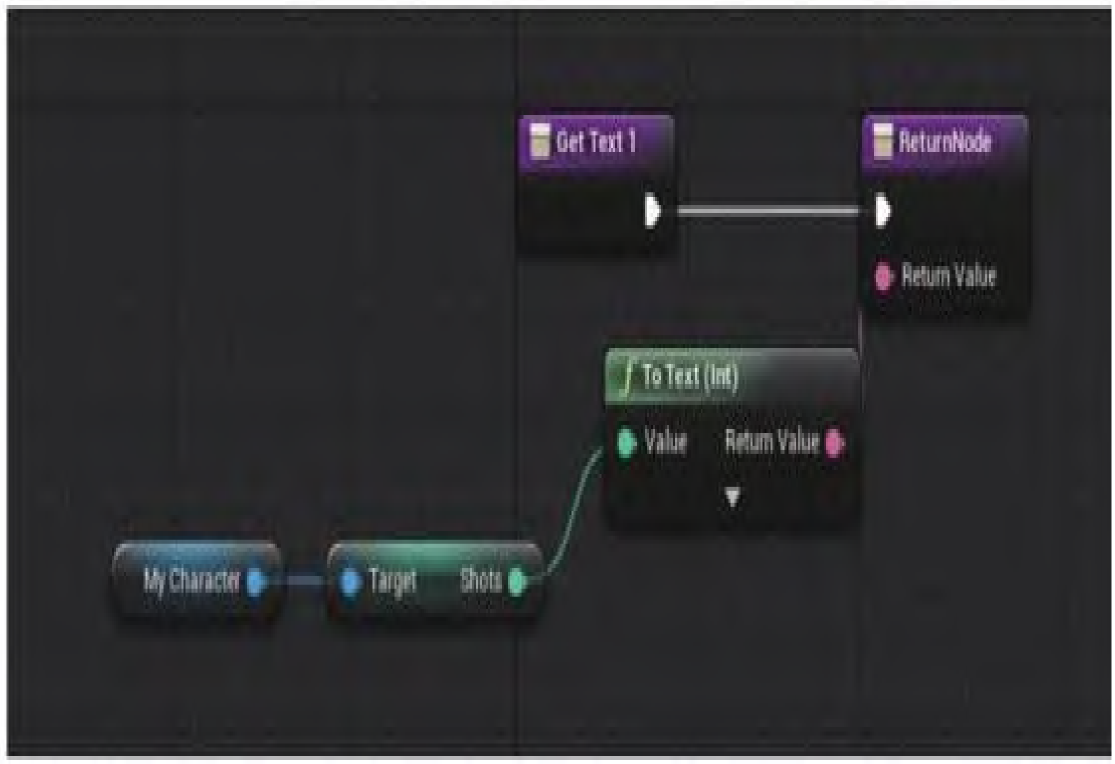



图4-81

这样将会更新并显示每次扣扳机时剩余的子弹数量。

**Step 9**　在Designer（设计）选项卡，单击设置Score的值，然后在详细信息面板，单击绑定选项并创建下方的绑定，如图4-82所示。

图4-82

最终，它会在撞击到岩石时为玩家获取并显示合适的分数。

**Step 10**　单击编译和保存按钮，然后关闭蓝图。

**Step 11**　在主编辑器视图，单击编辑器中的Play按钮以播放，如图4-83所示。

图4-83

行了，你的简单的射击场游戏已经完成了。在这个指南中，我们学习了如何使用Event Dispatcher（事件调度器）、Cast（投射）节点和Binding（绑定）来在三种不同类型的蓝图间进行通信。我们学习了如何从关卡向角色蓝图传递信息，以及如何从角色向HUD进行变量通信。

在最后一步，你将会看到一些用来变更此项目的示例，你可以使用它们来完成自己的项目。我们会给你提供一些链接，供你访问文档，从而了解本指南中的主题。

### 4.2.8　发挥想象

如果你想使用好在学习本项目的过程中学到的内容，请尝试进行如下操作：

•在FirstPersonCharacter蓝图中，对首次、第二次或第三次击中岩石给予更高的分数。

•同时在FirstPersonCharacter蓝图中，将其设置为除非生成岩石，否则玩家无法进行射击。

•在关卡中，再添加两个生成点（可能在角落，或在玩家的左/右侧）。

•在HUD中，添加新分类来追溯由玩家击中的“首次射击”数量，如图4-84所示。

图4-84

4.3

# 管理蓝图通信

## 4.3.1　概述

当你应用蓝图时，有时你可能会发现需要访问另一个蓝图中包含的函数、事件及变量。最简单的方法是通过一个公开暴露的对象变量来引用“目标”蓝图，然后指出你想使用的蓝图实例。

## 4.3.2　引用目标蓝图的简单方法

下面给出了一个示例，展示了从你正在处理的蓝图中引用一个目标蓝图的最简单、最通用的方法。

### 4.3.2.1　创建目标蓝图的对象变量

当你正在应用一个蓝图时，创建一个引用目标蓝图的目标变量可以使你访问该蓝图的函数、事件及变量。

要想创建一个引用目标蓝图的变量：

**Step 1**　在蓝图编辑器中打开你正在处理的蓝图（正在工作中的蓝图）。

**Step 2**　在图表模式中，通过在我的蓝图面板中单击Variable（变量）按钮来创建一个新变量，如图4-85所示。

**Step 3**　给这个新变量取一个和目标蓝图相关的名称，比如“TargetBlueprint”。

**Step 4** 在详细信息面板中，单击Variable Type（变量类型）旁边的下拉框。展开Object（对象）部分，如图4-86所示。

图4-85

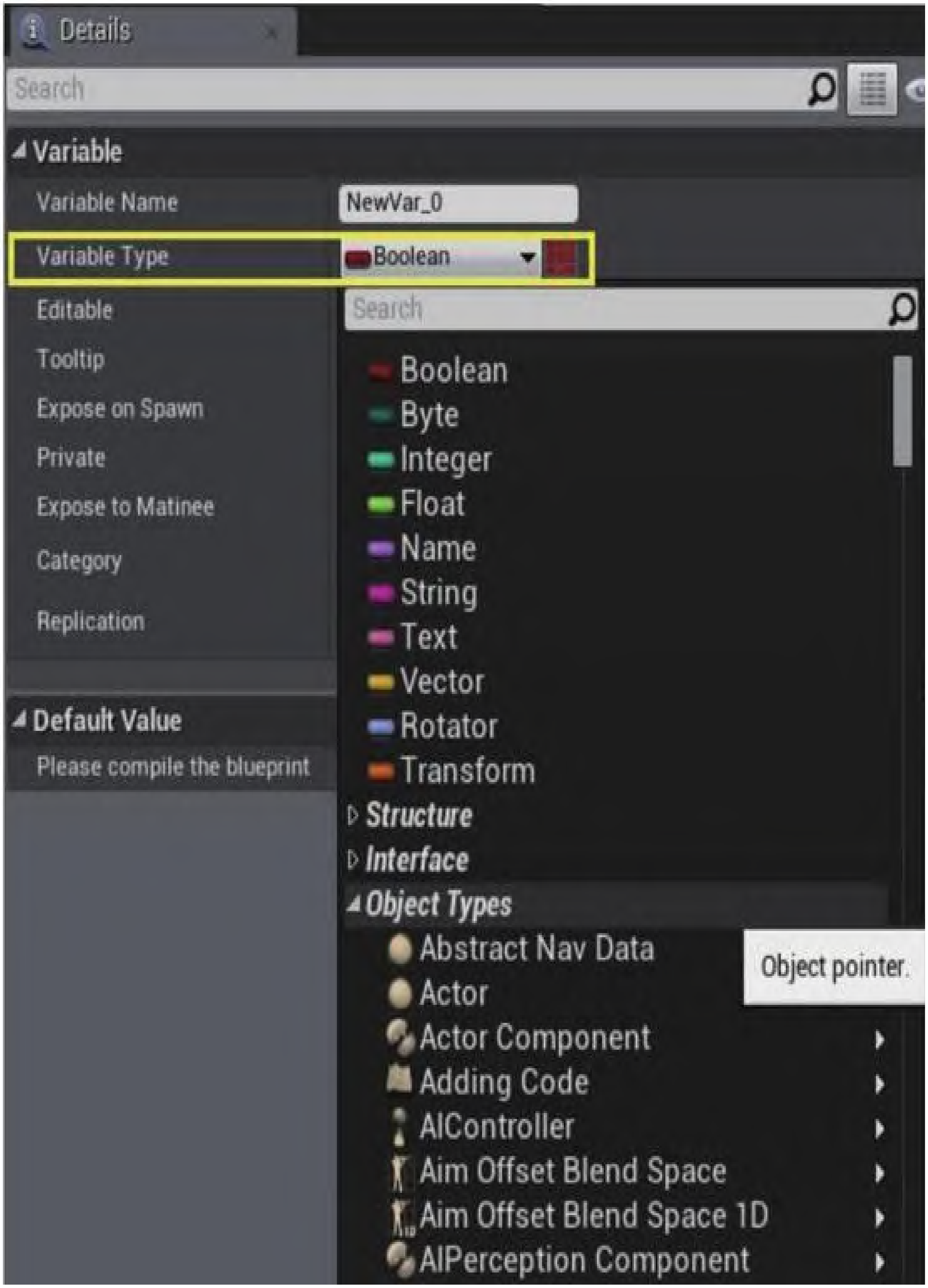



图4-86

**Step 5**　滚动列表，并选择你想引用的那个蓝图的名称。蓝图类型的引用在其名称后会附加“_C”，如图4-87所示。

Step 6　作为最佳实践，最好给你的新对象变量添加一个工具提示信息，可以通过在Tooltip（工具提示信息）框中输入文本来完成，比如“到你想通信的蓝图的引用”，如图4-88所示。

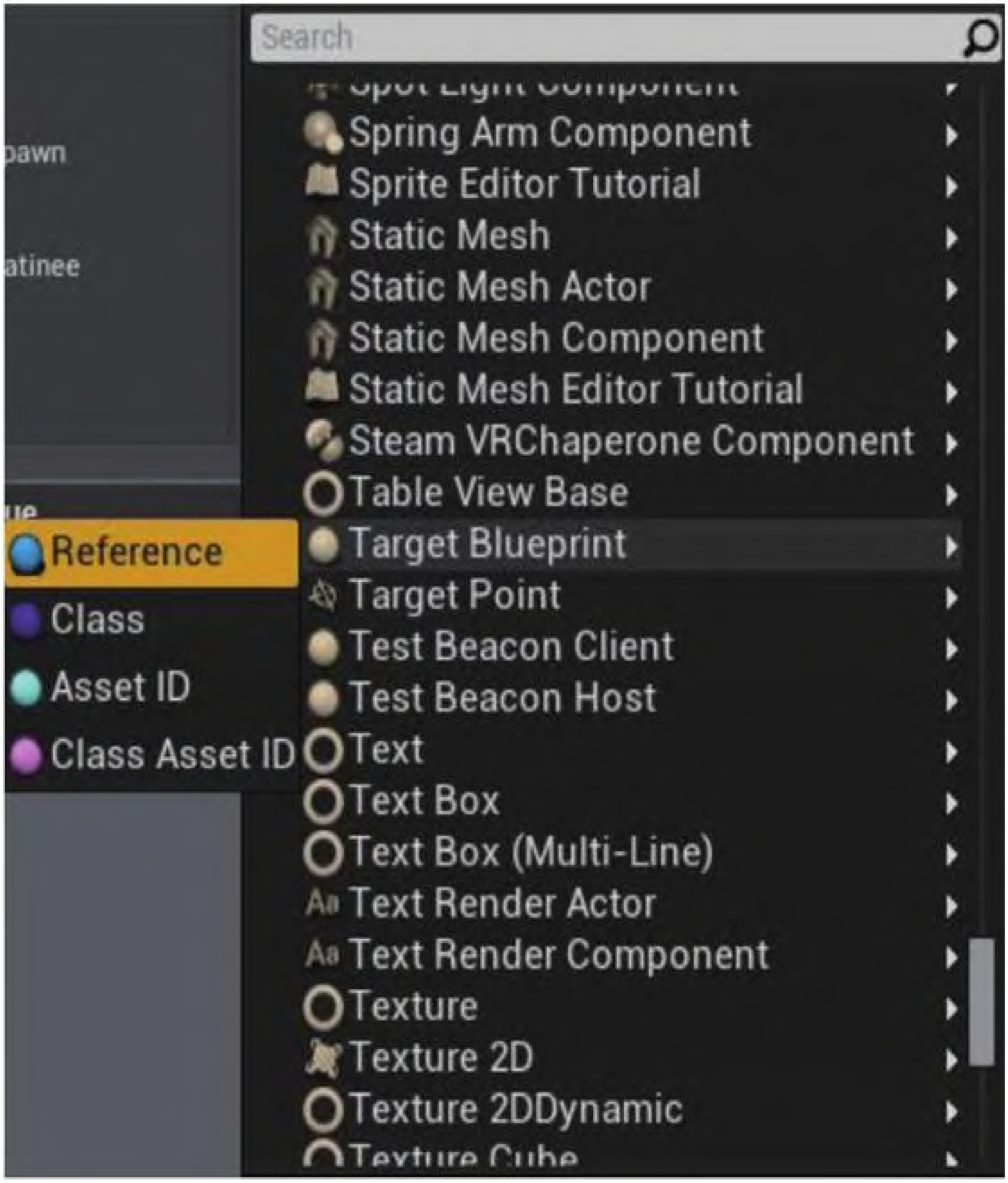



图4-87

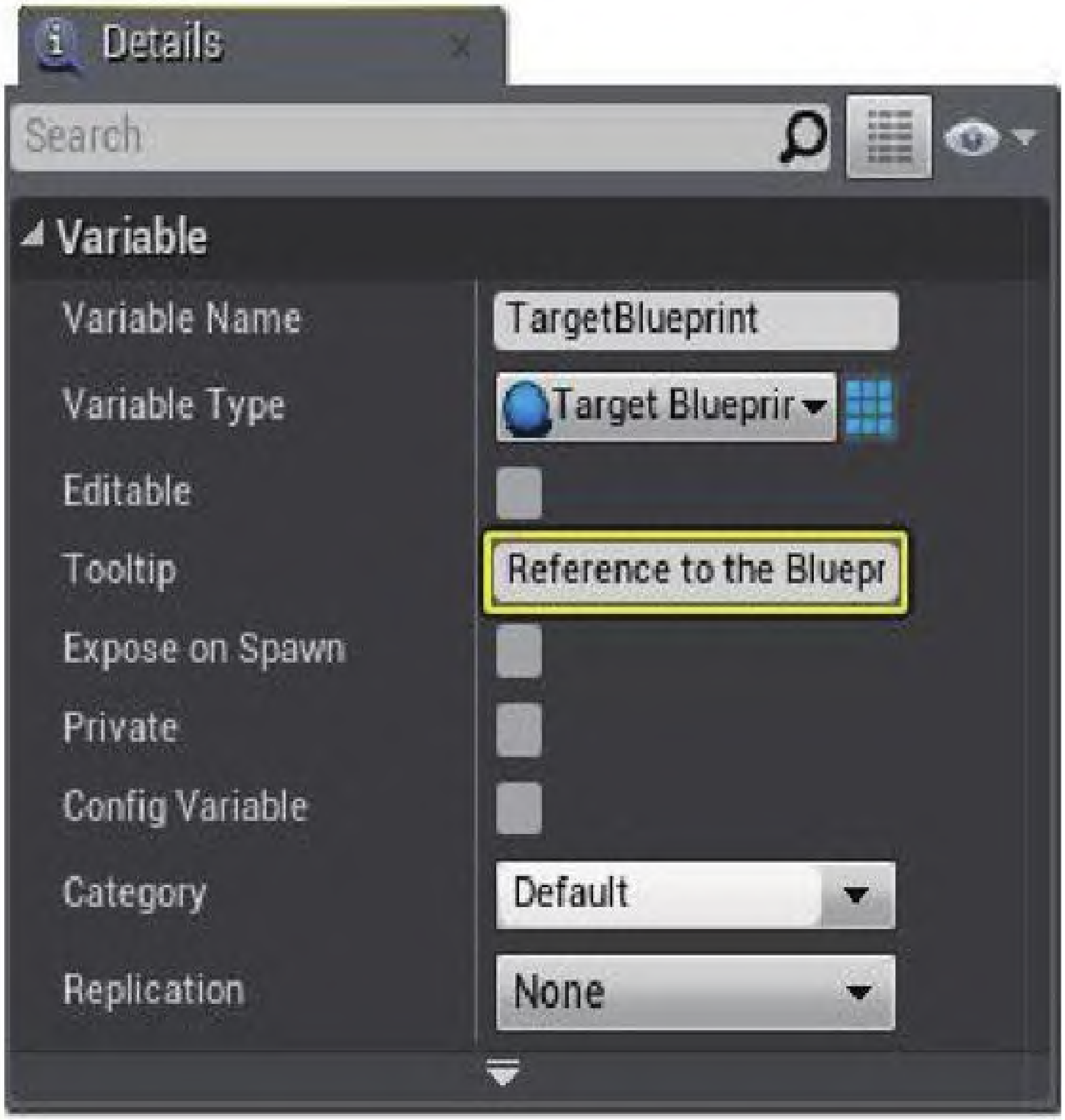



图4-88

**Step 7** 在我的蓝图面板中，单击新对象变量旁边的“闭合眼睛”图标，把它变为“睁开的眼睛”。这通过使该对象变量变为公有状态来暴露该变量，从而可以在关卡编辑器的详细信息面板中设置它们的值，如图4-89和图4-90所示。

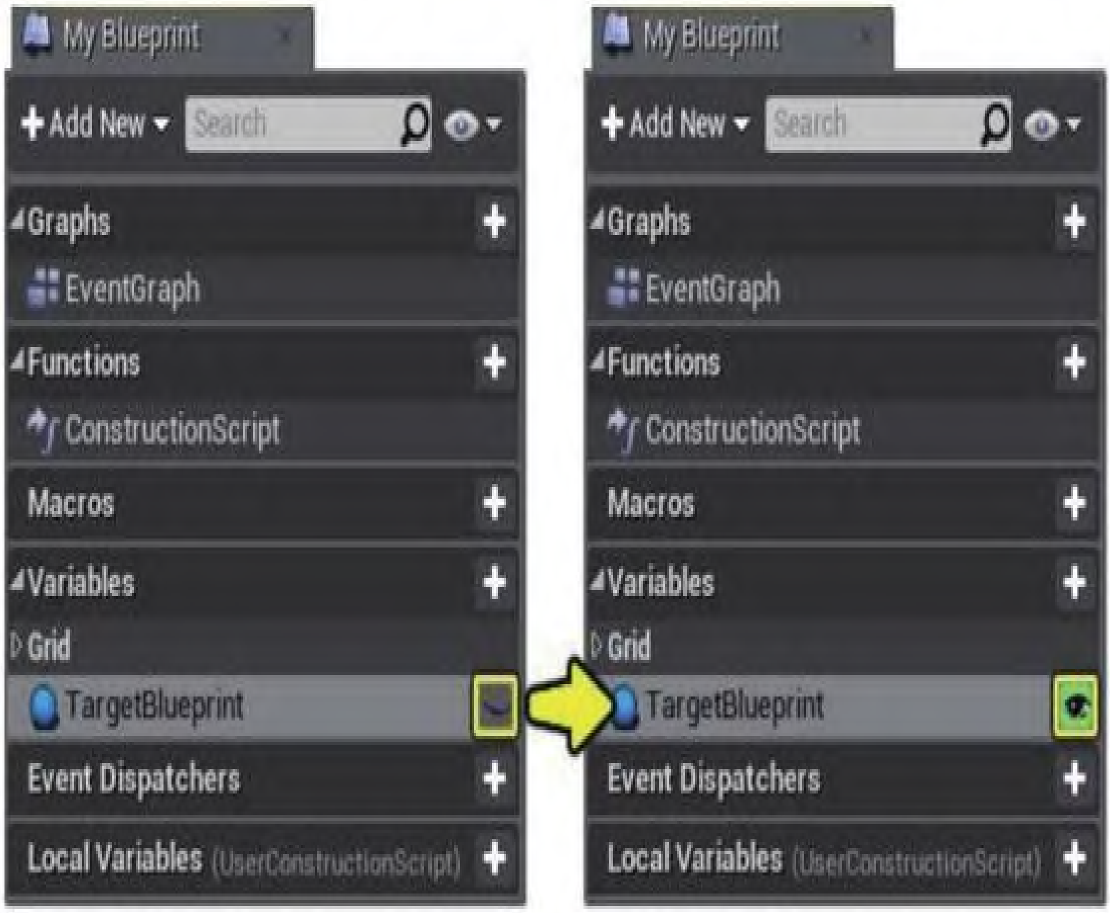

My Blueprint
Add New
Search
Graphs
EventGraph
Functions
ConstructionScript
Macros
Variables
Grid
TargetBlueprint
Event Dispatchers
Local Variables (UserConstructionScript)
My Blueprint
Add New
Search
Graphs
EventGraph
Functions
ConstructionScript
Macros
Variables
Grid
TargetBlueprint
Event Dispatchers
Local Variables (UserConstructionScript)

图4-89

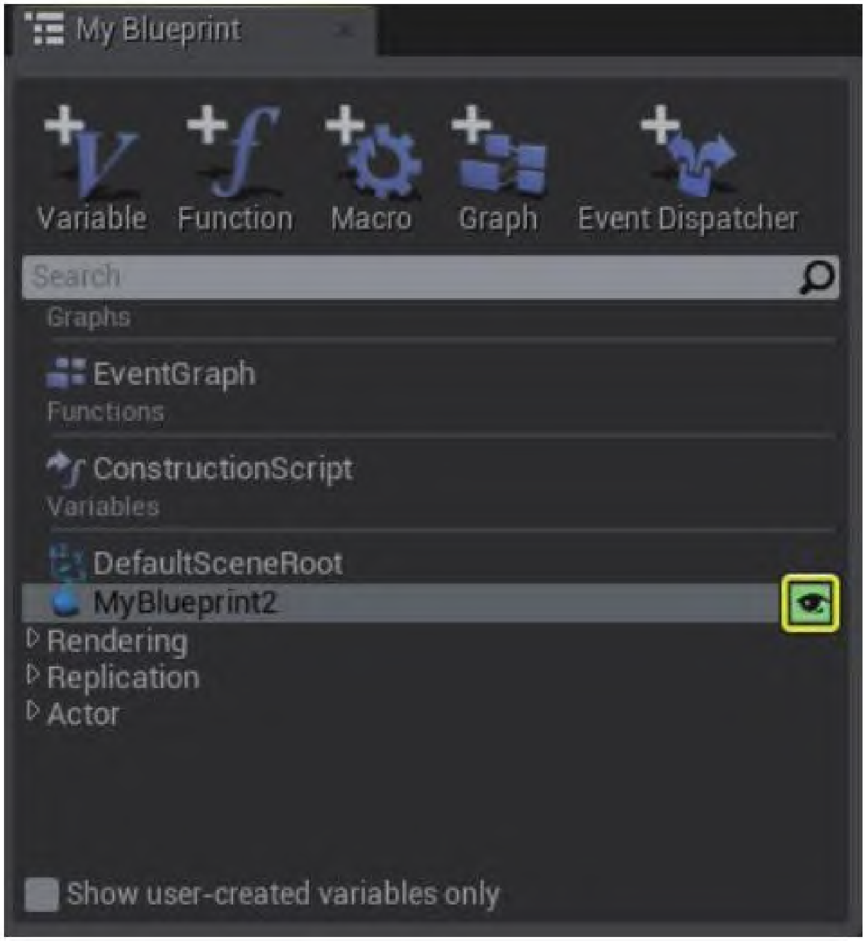



图4-90

“睁开的眼睛”图表周围的绿色表示该暴露的变量有工具提示信息。如果该暴露的变量没有工具提示信息，那么眼睛周围的区域将是黄色。

Step 8　编译并保存蓝图。

在决定暴露哪个变量、函数及事件时一定要小心处理。除非其他人访问及修改这些变量是安全的，否则不要暴露这些变量。作为最佳实践指南，最好仅暴露其他蓝图需要的变量及你想让关卡设计师修改的变量，而不是暴露所有内容。

#### 4.3.2.2　指定蓝图实例

类蓝图可以作为一种模板或原型，你可以根据它来创建多个实例。由于这个原因，所以当你正在引用一个目标蓝图时，你可以指定你想使用的目标蓝图实例。

要想指定你想使用的目标蓝图实例：

**Step 1**　在关卡编辑器的详细信息面板中，在选中正在工作的蓝图的情况下，查找你刚刚创建的公开暴露的变量的部分。如果你没有在创建变量时指定不同的部分，那么你可以在Default（默认）部分找到它（注意，你必须在你的关卡中放置一个目标蓝图的实例，以便可以在详细信息面板中选择一个蓝图）。

**Step 2**　在引用目标蓝图的对象变量名称旁边的下拉框中，选择你想使用的蓝图实例，如图4-91所示。

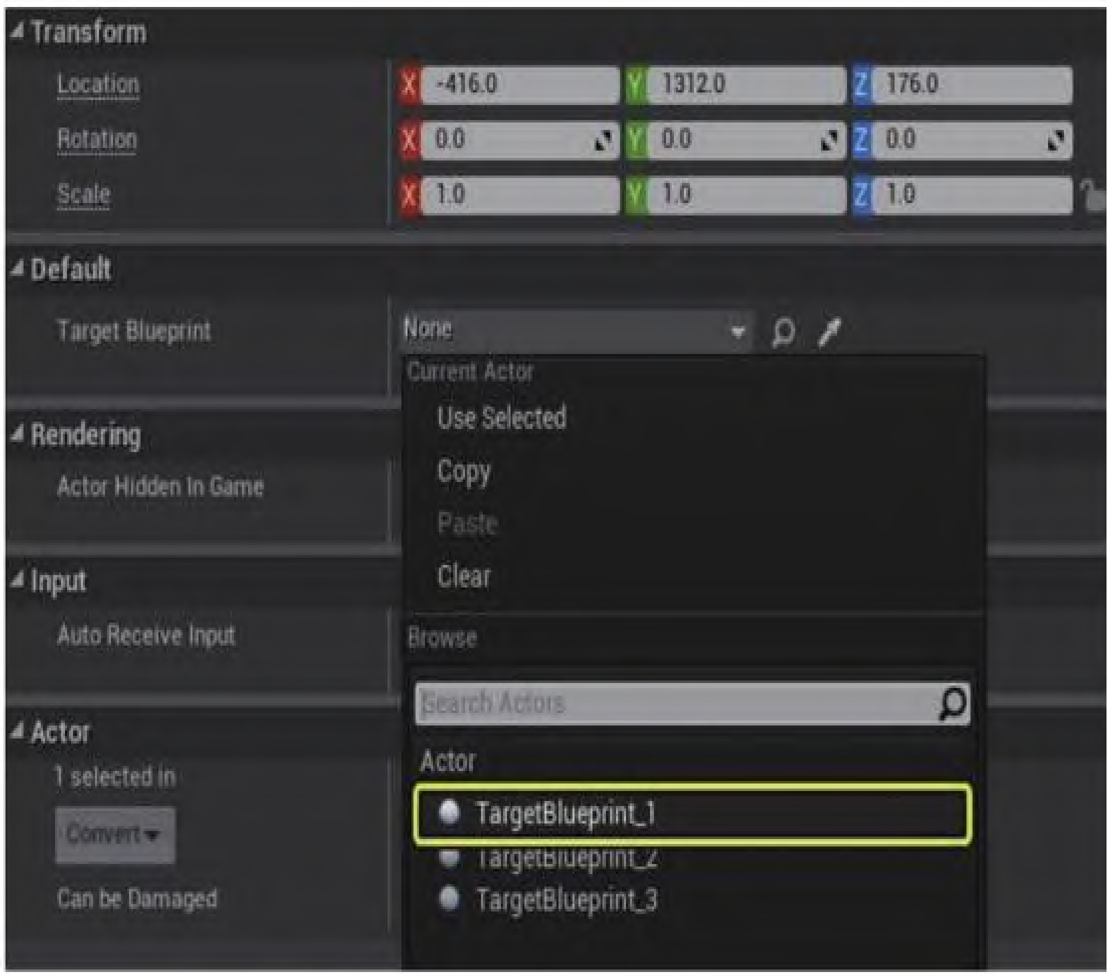



图4-91

如果你正在处理的蓝图是关卡蓝图，那么你应该在蓝图编辑器中指定你要使用的蓝图实例，而不是在关卡编辑器中。默认情况下，蓝图编辑的详细信息面板位于左下角。

如果你不确定列表中的哪个实例和你想使用的实例相对应，那么你可以在场景大纲视图或者视口中选择你想使用的实例，然后在下拉菜单中单击Use Selected（应用选中项）。

#### 4.3.2.3 使用目标蓝图的函数、事件及变量

在你为目标蓝图创建了一个对象变量后，你可以通过按住Ctrl键并拖动那个变量到图表中来获得到这个变量的引用，这会为它创建一个变量“getter”节点。然后，若你从那个节点的输出引脚开始向外拖动鼠标，将会出现关联菜单，显示了你可以使用的目标蓝图中的事件、函数及变量的列表。现在你可以像使用其他任何脚本元素一样使用这些元素了，如图4-92所示。

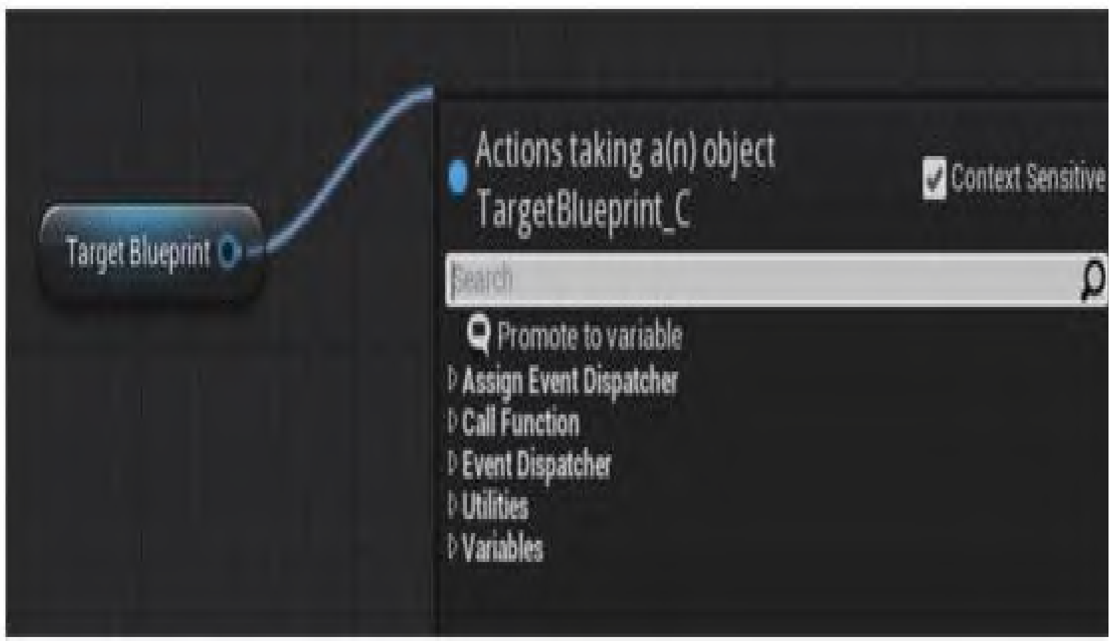



图4-92

### 4.3.3 引用目标蓝图的其他方法

从正在工作的蓝图中引用目标蓝图的更复杂的方法有：

•将类型转换为目标蓝图类型。

•指定目标蓝图类型作为一个函数的参数。

#### 4.3.3.1 将类型转换为蓝图类型

你可以将类型转换为蓝图类型，就像你将类型转换为其他Actor一样。

要想将类型转换为一个蓝图类型：

**Step 1** 在事件图表中，从一个节点的输出引脚拖动鼠标，该节点的关联菜单中要包含Casting（类型转换）节点，比如Actor的对象变量。在出现的关联菜单中，开始输入“cast”。滚动列表到Casting（类型转换）下来查找你的目标蓝图类型，并选择它，如图4-93和4-94所示。

图4-93

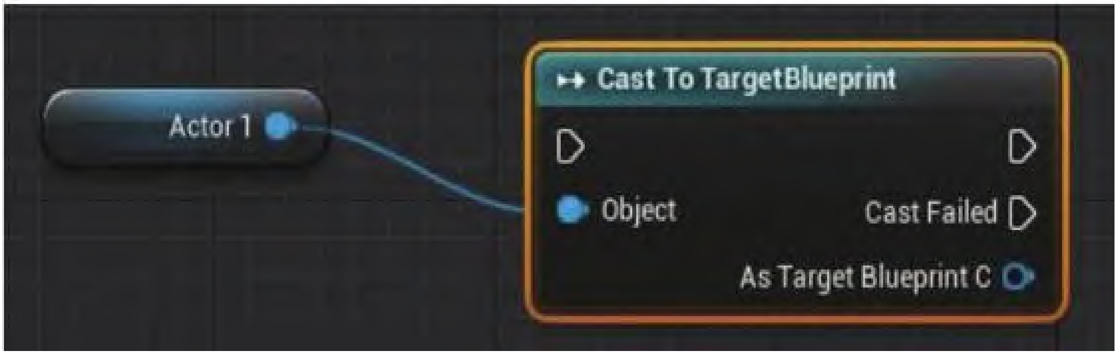



图4-94

Step 2　从列出了目标蓝图类型的输出引脚拖动鼠标来查看关联菜单，该菜单中包含了目标蓝图的函数、事件及变量，如图4-95所示。

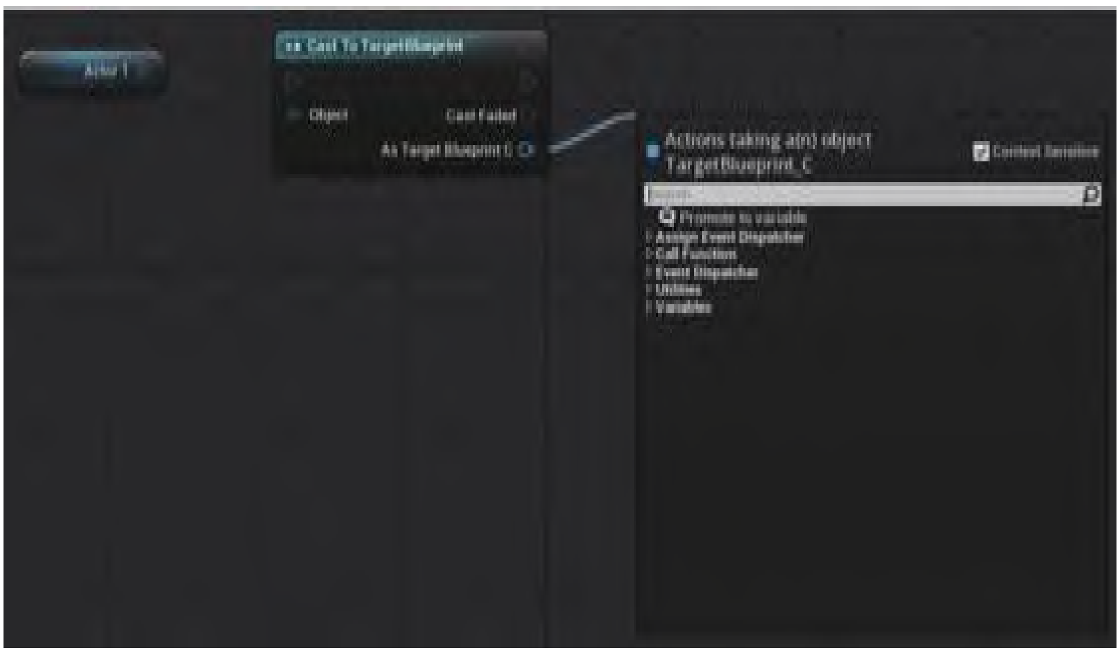



图4-95

如果类型转换成功，那么你可以使用目标蓝图类型中的任何函数、事件及变量。你从这个关联菜单中添加的任何节点都将会自动地连接到该类型转换节点的success输出执行引脚，如图4-96所示。

图4-96

#### 4.3.3.2 指定一种蓝图类型作为函数参数

当你在蓝图中应用函数时，你可以把目标蓝图类型指定为那个函数的参数。

要想创建一个指定蓝图类型作为函数参数的函数：

**Step 1** 在正在工作的蓝图内，从我的蓝图面板中，单击Function（函数）按钮来创建一个函数，如图4-97所示。

**Step 2** 在详细信息面板中，通过单击Function（新建）按钮来添加一个输入，如图4-98所示。

图4-97

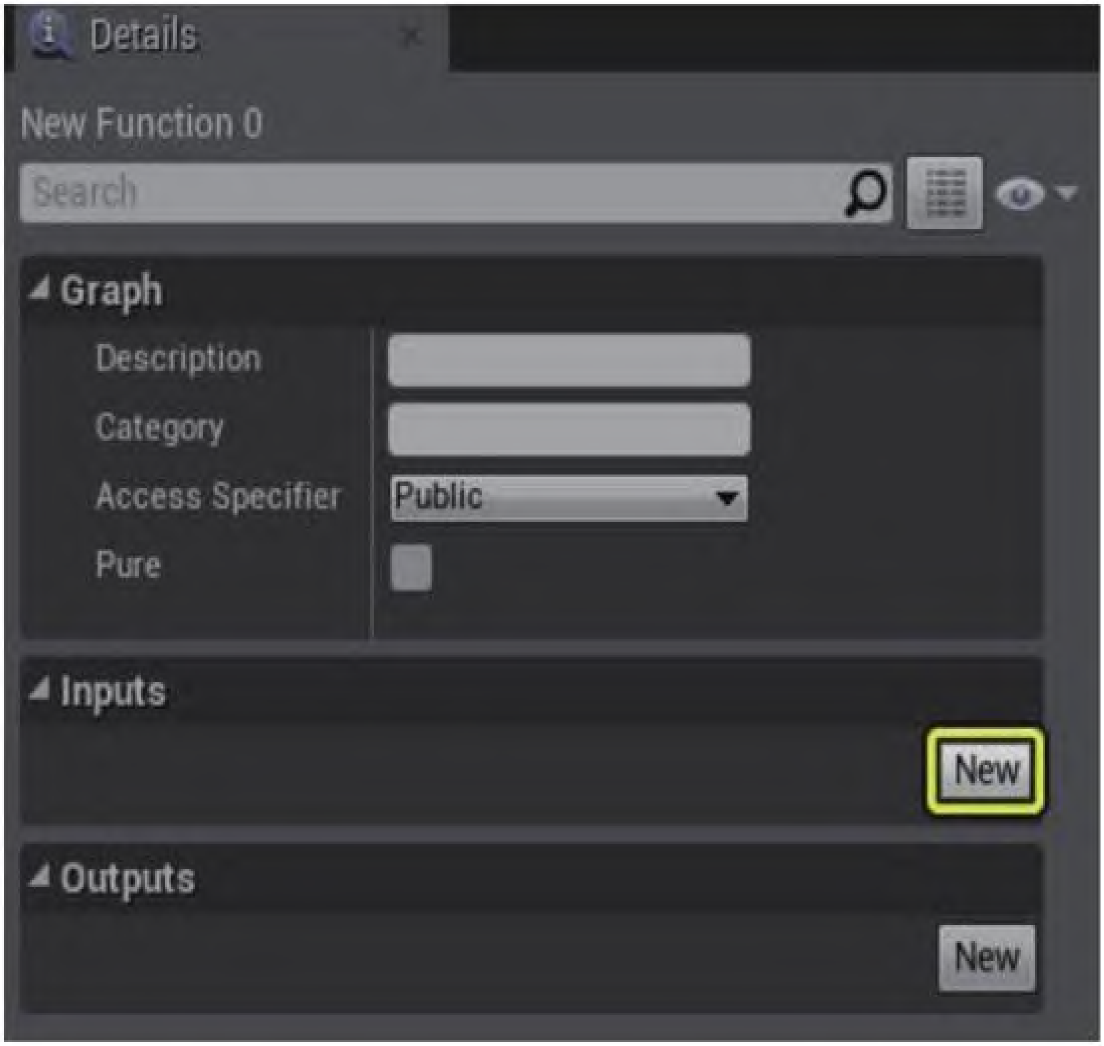



图4-98

**Step 3**　在NewParam（新建变量）旁边的下拉框中，展开Object（对象）部分，找到目标蓝图类型，并选择它，如图4-99所示。

**Step 4**　在事件图表中，从新变量的输出引脚开始拖动鼠标。出现的关联菜单包含了目标蓝图的函数、事件及变量，现在你就可以使用它们了，如图4-100所示。

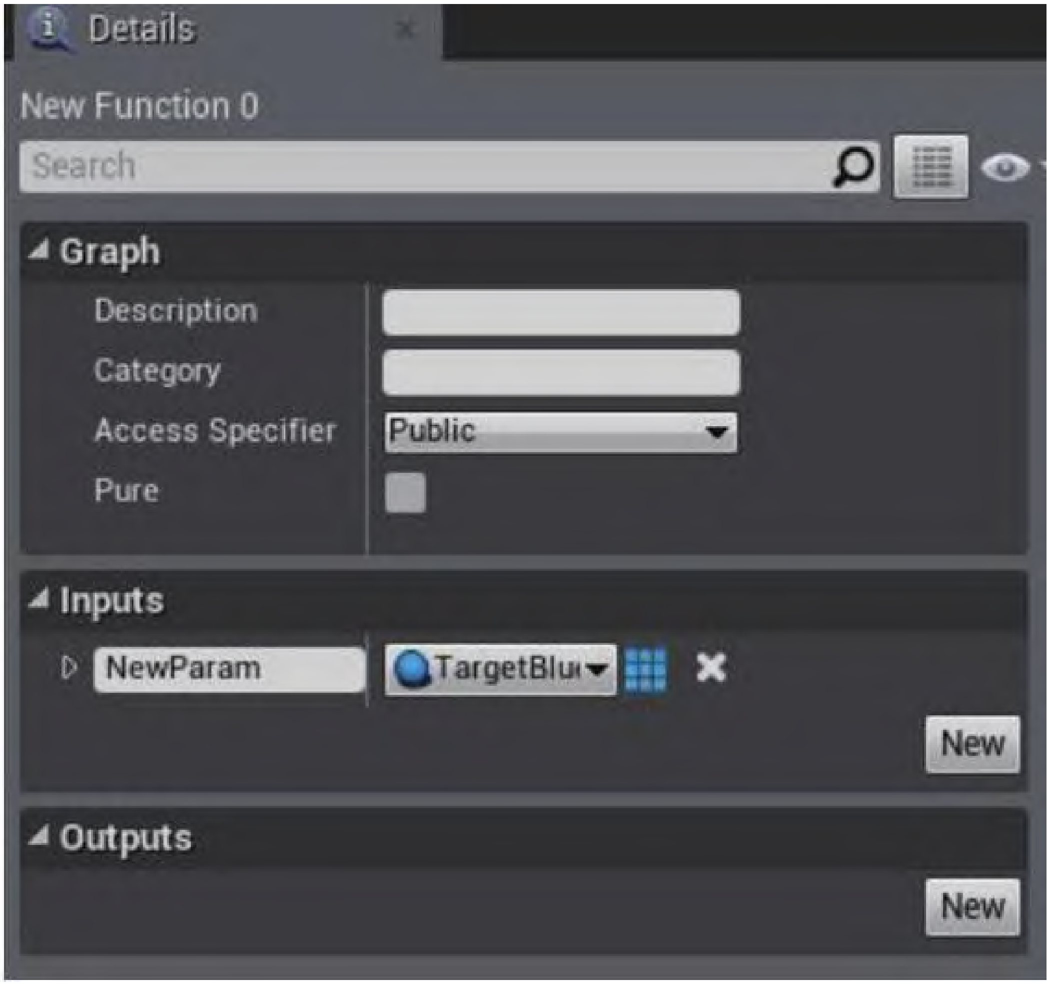

Details
New Function 0
Search
Graph
Description
Category
Access Specifier
Public
Pure
Inputs
NewParam
TargetBlu
New
Outputs
New

图4-99

图4-100

## 4.3.4 问题解决

一个常见的错误是你的变量中没有指定实例。如果你正在从当前工作的蓝图中调用目标蓝图的脚本时该脚本不能工作，那么请确保你在当前工作的蓝图的详细信息面板中已经指定了你想使用的蓝图实例。但是，如果你通过函数调用的结果设置了该变量，那么你应该会在你的日志中看到一个警告，提示你的变量有一个“Access None（没有访问项）”异常。如果你调出信息日志窗口，它应该会提供了一个到导致该问题的节点的可单击链接。

## 4.3.5 蓝图接口迁移

### 1. 概述

如果你正在使用Blueprint Interface（蓝图接口）来让蓝图之间彼此通信，那么你可以轻松地把蓝图接口中的函数转换成正常的蓝图函数，以便你的蓝图间可以直接通信。

### 2. 迁移过程

要想从一个蓝图接口中进行迁移：

Step 1　打开引用该蓝图接口的蓝图。

**Step 2**　在蓝图编辑器中，单击工具条中的Blueprint Props（蓝图属性）按钮，来在详细信息面板中显示蓝图的属性，如图4-101所示。

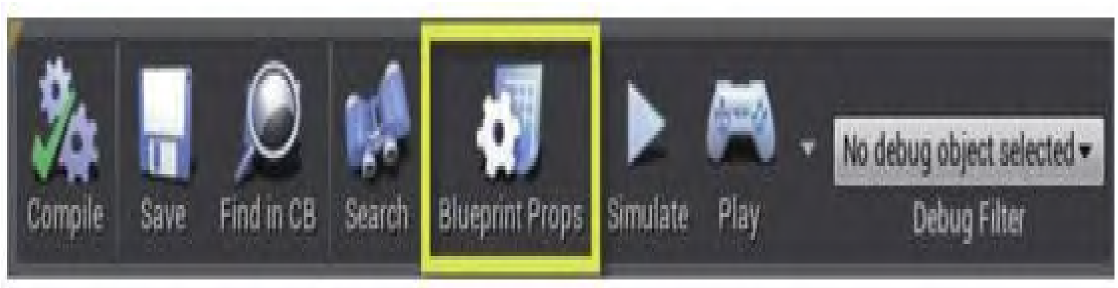



图4-101

**Step 3**　在详细信息面板中的Interfaces（接口）下，找到你想转换的蓝图接口。单击其名称旁边的白色“X”图标，如图4-102所示。

**Step 4**　在出现的对话框中，单击“是”，如图4-103所示。

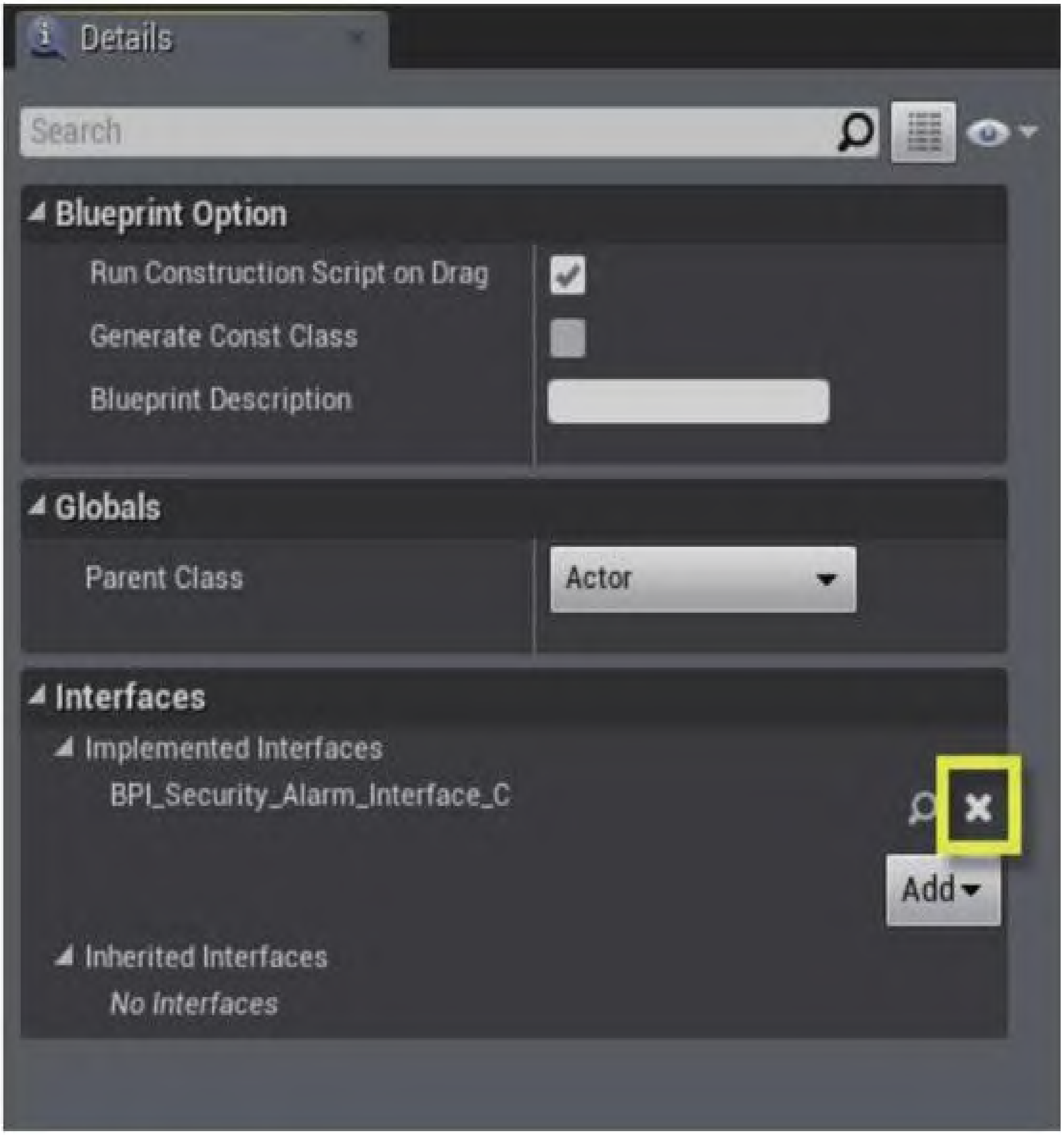

Details
Search
Blueprint Option
Run Construction Script on Drag
Generate Const Class
Blueprint Description
Globals
Parent Class
Actor
Interfaces
Implemented Interfaces
BPI_Security_Alarm_Interface_C
Add
Inherited Interfaces
No Interfaces

图4-102

Message
Would you like to transfer the interface functions to be part of your blueprint?
Copy Message
Yes
No

图4-103

如果你单击Function（函数）按钮来添加蓝图接口中的函数，那么它们会被转换为正常的蓝图函数。现在你可以像应用其他任何蓝图函数一样来应用它们了。

### 3. 在蓝图中查找

在蓝图编辑器中，你可以使用“Find in Blueprints（在蓝图中查找）”功能（在蓝图中查找功能在编辑菜单上，或者按下Ctrl+F组合键调出该功能），来查找你应用的所有接口信息及所有蓝图接口实现，这会使得迁移变得更加简单。

当使用“在蓝图中查找”功能时，通过取消选中Find In Current Blueprint Only（仅在当前蓝图中查找）复选框，你可以同时在所有蓝图中查找你正在搜索的接口信息，如图4-104所示。

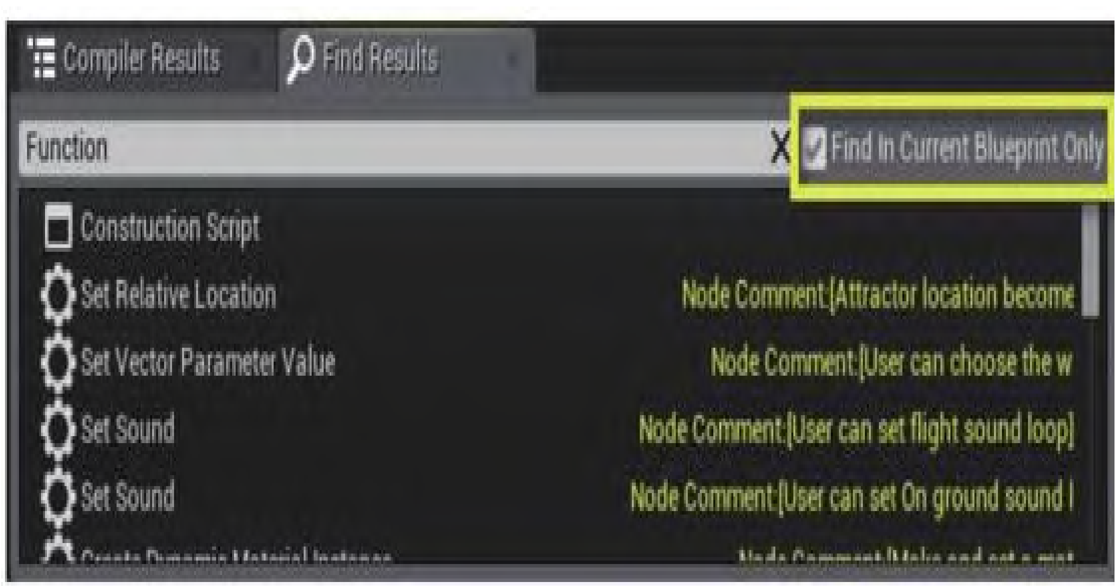



图4-104

4.4

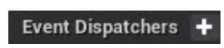
# 事件调度器

通过向Event Dispatcher（事件调度器）绑定一个或多个事件，你可以在调用该事件调度器后触发所有事件。这些事件可以在类蓝图中进行绑定，但事件调度器也允许在关卡蓝图激活这些事件。

**1. 创建事件调度器**

事件调度器是在Graph Mode中我的蓝图选卡内进行创建的。

要想创建一个新的事件调度器：

**Step 1** 单击我的蓝图选卡顶部的 

按钮。

**Step 2** 在我的蓝图选卡中的列表尾部出现的名称文本域中，输入该事件调度器的名称，如图4-105所示。

图4-105

**2. 设置属性**

通过在我的蓝图选卡内选中该事件调度器，你可以在详细信息面板中编辑其属性。你可以为你的事件调度器设置工具提示信息和类目，且可以添加输入。

给你的事件调度器添加输入，允许你向绑定到你的事件调度器的每个事件发送变量。这使得数据不仅可以在你的类蓝图中流动，且可以在你的类蓝图和关卡蓝图之间流动。

给你的事件调度器添加输入的过程和给函数、自定义事件及宏添加输入及输出的流程类似。如果你想使用和另一个事件相同的输入，那么你可以使用Copy Signature from（复制签名）下拉菜单来指定该事件。要想给该事件调度器添加你自己的输入：

**Step 1** 单击详细信息面板的Inputs（输入）部分的New（新建）按钮，如图4-106所示。

**Step 2** 命名该新输入并使用下拉菜单设置其类型。在这个示例中，有一个String（字符串型）的输入参数MyStringParam，如图4-107所示。

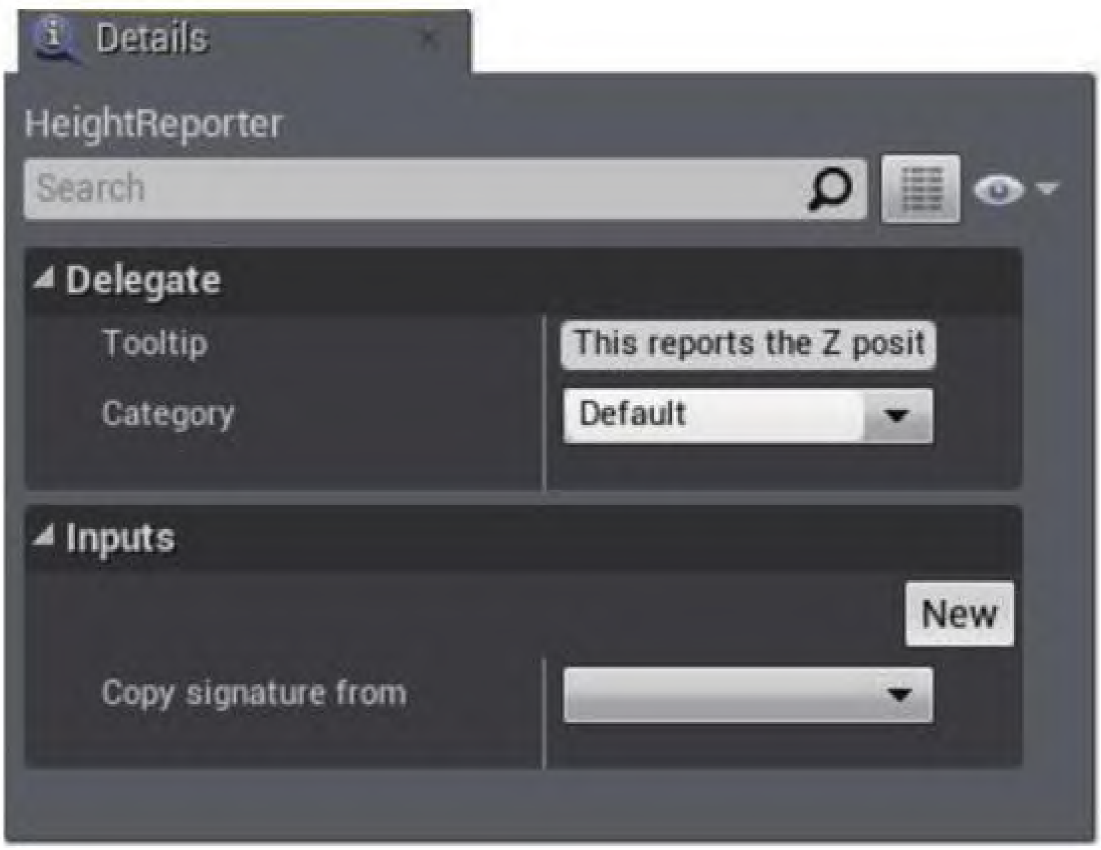



图4-106

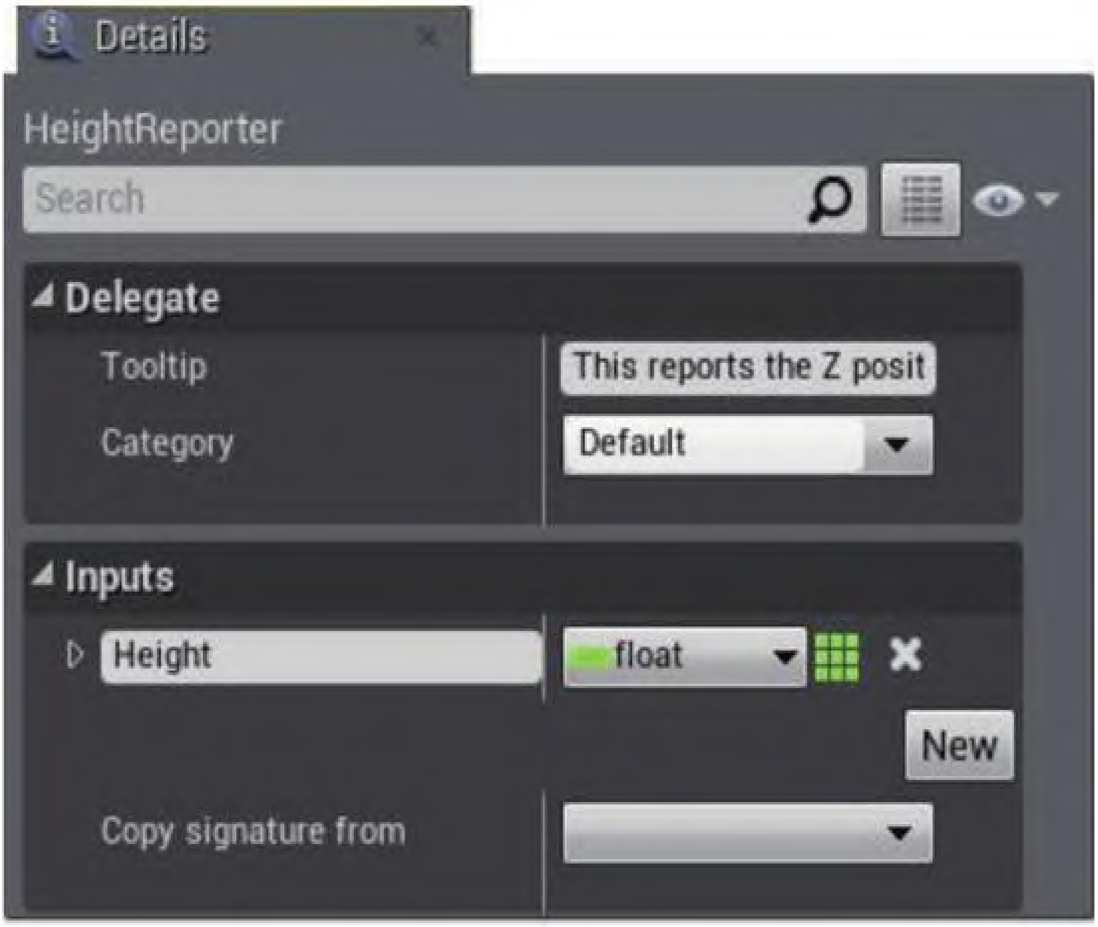



图4-107

你可以展开该参数项来设置默认值，并指出是否通过引用传递参数，如图4-108所示。

要想改变节点边缘上的这个参数的引脚的位置，请使用展开的详细信息面板项的向下和向上箭头。

### 3. 使用事件调度器

一旦你创建了事件调度器后，你可以添加事件节点、绑定节点及解除绑定连接到其上面的节点。尽管你可以在我的蓝图选卡中双击事件调度器选项来打开事件调度器的图表，但是该图表是锁定的，你不能直接修改该事件调度器。绑定、解除绑定及分配方法都允许你向事件调度器的事件列表添加事件，但调用方法将激活事件图表中存储的所有事件。

以下描述的所有节点，都可以添加到类蓝图及关卡蓝图中。除了Event（事件）节点外，每个节点都有一个Target（目标）输入引脚：

•在类蓝图中，该引脚自动设置为Self。这意味着对于该类来说该事件列表是变化的，所以每个类实例都将会发生变化。

•在关卡蓝图中，这个引脚必须连接到关卡中的一个类实例的引用上。这意味着事件列表将仅会针对那个类的特定实例进行改变。关卡蓝图文档解释了如何创建你需要的任何Actor引用的过程。

4. **调用事件调度器**

使用Call节点调用一个Event Dispatcher（事件调度器），将会触发绑定到该事件调度器的所有事件。对于每个事件调度器，你可以应用多个Call节点，且你既可以在类蓝图中调用事件调度器也可以在关卡蓝图中进行调用，如图4-109所示。

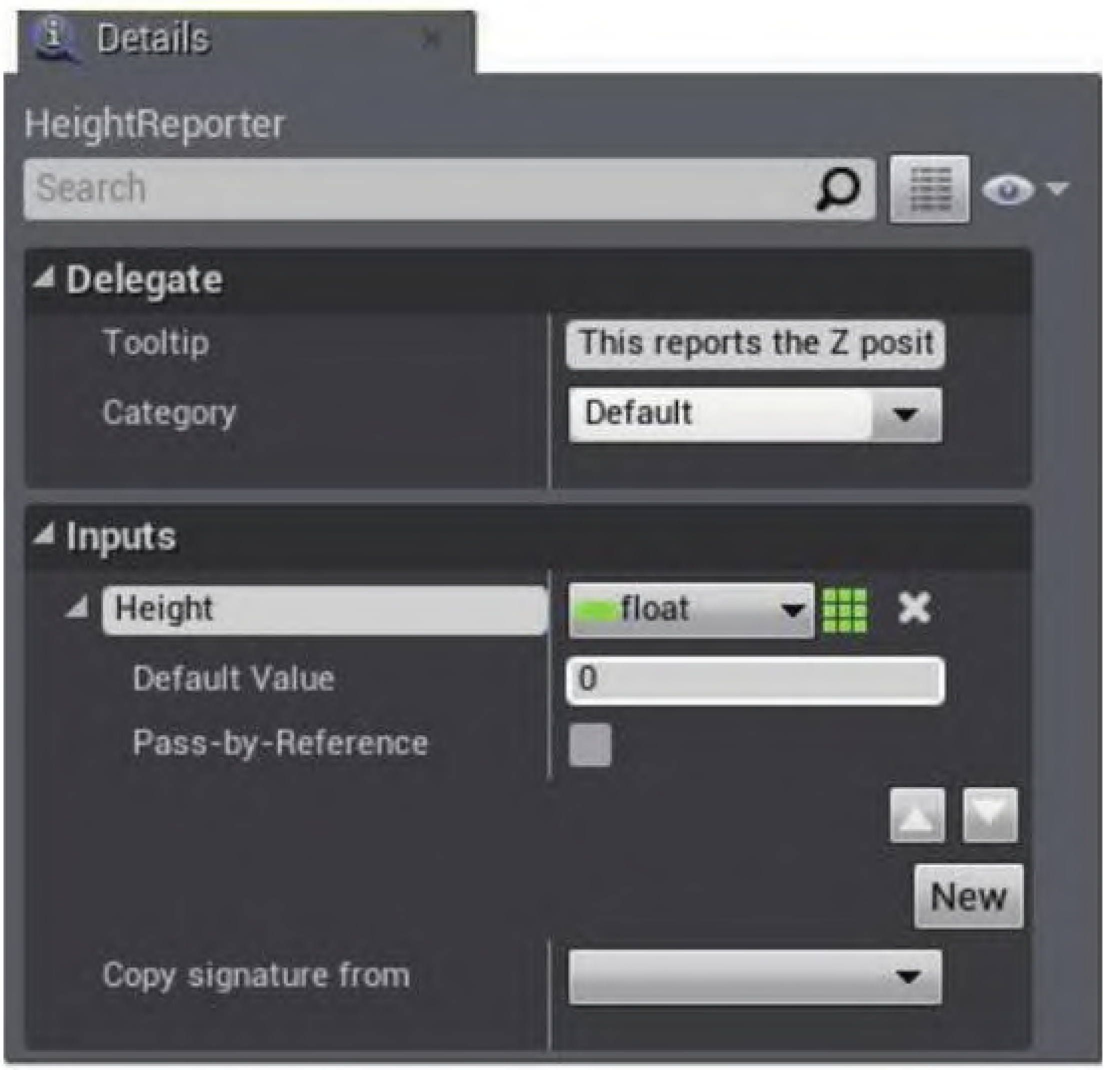



图4-108

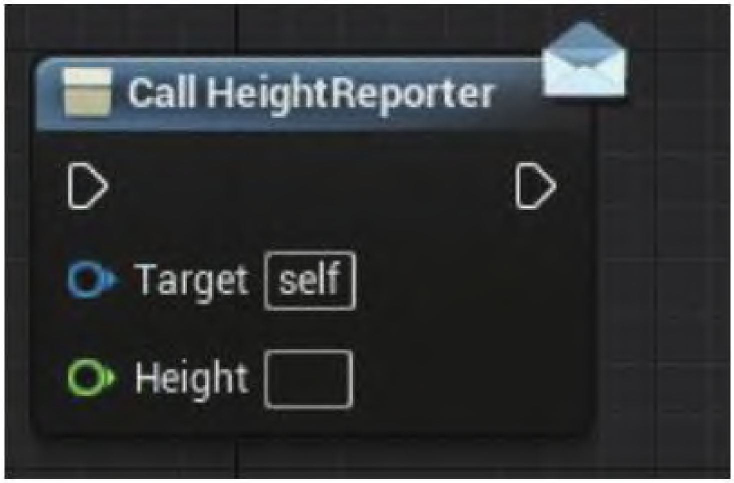



图4-109

5. 绑定及解除绑定事件

如果没有事件绑定到事件调度器，那么调用该事件调度器将不会产生任何效果。鉴于每个事件调度器都有一个和其相关的事件列表。向该列表中添加一个事件的方法是使用Bind Event（绑定事件）节点，从该列表中删除事件的方式是使用Unbind Event（解除绑定事件）节点。另外，通过使用Unbind All Events（接触绑定所有事件），还可以解除当前绑定到事件调度器上的所有事件，如图4-110所示。

6. 创建调度器事件

通过使用Event Dispatcher（事件调度器）菜单上的Event（事件）选项创建一个具有正确签名的自定义事件，使得该事件可以绑定到该事件调度器上。尽管这个事件节点的看上去和没有连接到Bind节点的自定义事件节点类似，但是当执行Call事件调度器名称节点时将永远不会触发该事件，如图4-111所示。

图4-110

图4-111

## 4.4.1　绑定及解除绑定事件

如果没有事件绑定到事件调度器，那么调用该事件调度器将不会产生任何效果。鉴于每个事件调度器都有一个和其相关的事件列表。向该列表中添加一个事件的方法是使用Bind Event（绑定事件）节点，从该列表中删除事件的方式是使用Unbind Event（解除绑定事件）节点。另外，通过使用Unbind All Events（接触绑定所有事件），还可以解除当前绑定到事件调度器上的所有事件，如图4-112所示。

图4-112

尽管绑定事件节点可以执行多次，但每个事件仅能绑定一次。同时，类蓝图和关卡蓝图中的事件都添加到同一个事件列表中，所以Unbind All Events（解除绑定所有事件）节点将会取消绑定类蓝图和关卡蓝图中的所有事件。

•在类蓝图中执行Unbind All Events（取消绑定所有事件）节点，将针对该类的所有实例，取消绑定类蓝图和关卡蓝图中的事件。

•在关卡蓝图中执行Unbind All Events（取消绑定所有事件）节点，将仅针对Target端提供的内容，取消绑定类蓝图和关卡蓝图中的事件。

下面介绍创建Bind、Unbind及Unbind All节点的方法。

创建Bind Event（绑定事件）、Unbind Event（取消绑定事件）及Unbind All Events（取消绑定所有事件）节点的过程类似。尽管以下步骤仅解释了如何创建Bind Event（绑定事件）节点，但是只需在适当的步骤选择适当的菜单项，就可以创建Unbind Event（取消绑定事件）和Unbind All Events（取消绑定所有事件）节点。

**1.在类蓝图中进行创建**

**Step 1** 在My Blueprint（我的蓝图）选卡下，从Event Dispatcher（事件调度器）名称处开始拖动鼠标并将其放置到你正处理的图表中。

**Step 2** 在出现的菜单中选择Bind（绑定），如图4-113所示。

另一种方法：

**Step 1** 在图表中右击。

图4-113

**Step 2** 在出现的关联菜单中展开Event Dispatcher（事件调度器），如图4-114所示。

**Step 3** 选择Event Dispatcher（事件调度器）下的绑定事件到（事件调度器名称），如图4-115所示。

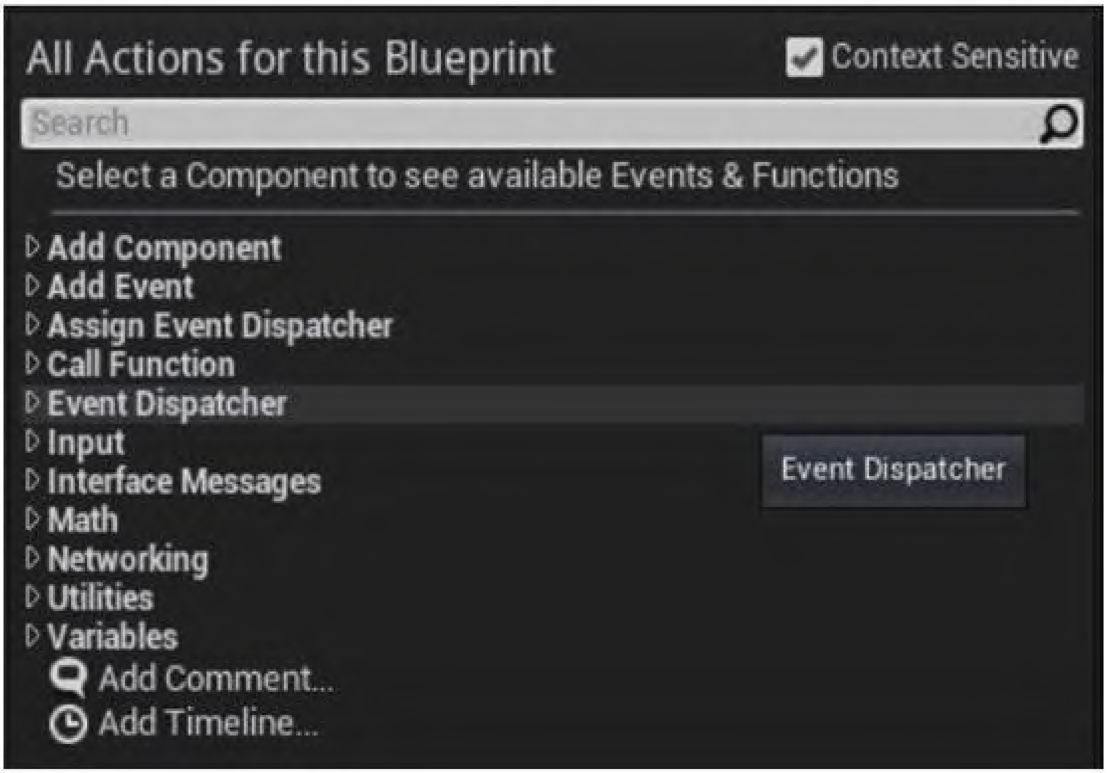

All Actions for this Blueprint
Context Sensitive
Search
Select a Component to see available Events & Functions
Add Component
Add Event
Assign Event Dispatcher
Call Function
Event Dispatcher
Input
Interface Messages
Math
Networking
Utilities
Variables
Add Comment...
Add Timeline...
Event Dispatcher

图4-114

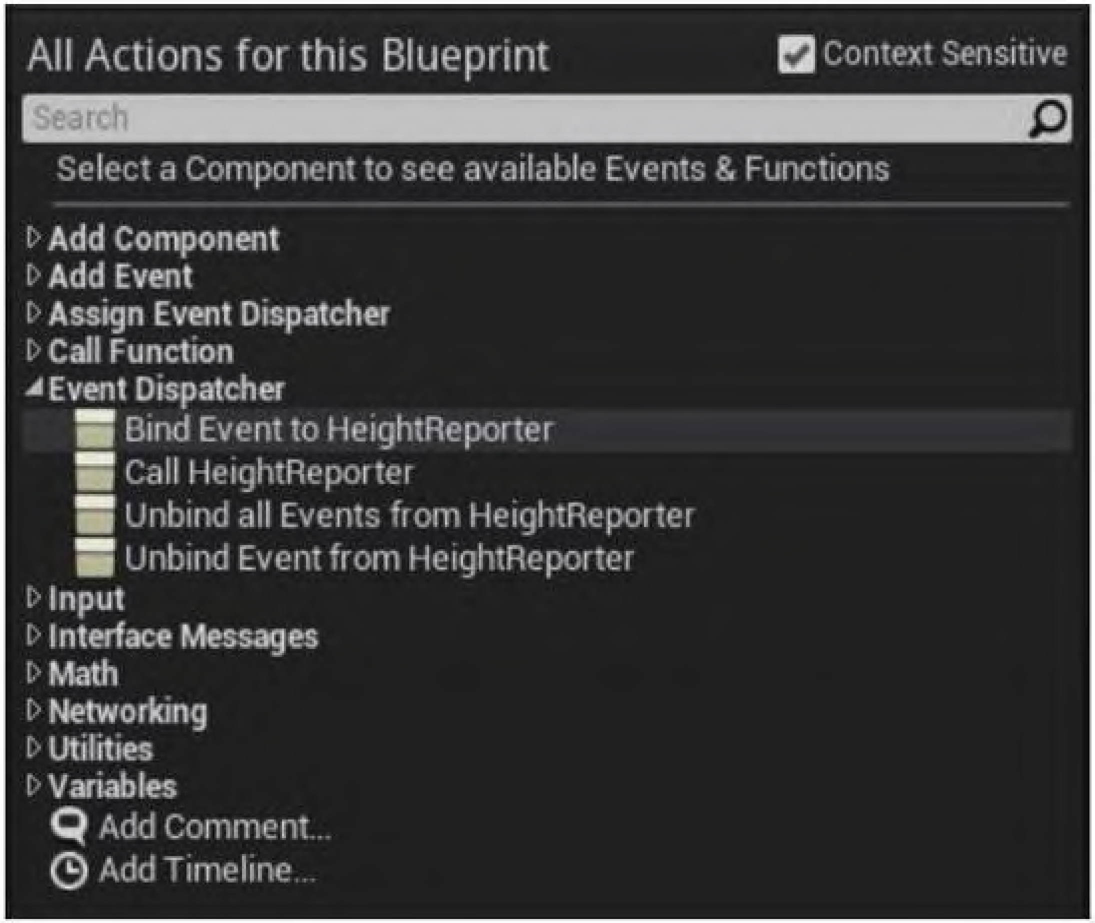



图4-115

### 2. 在关卡蓝图中进行创建

**Step 1** 添加到关卡中你想为其调用事件调度器的Actor的引用，如图4-116所示。

**Step 2** 从该引用节点的输出引脚开始拖动鼠标，然后释放鼠标来显示关联菜单，如图4-117所示。

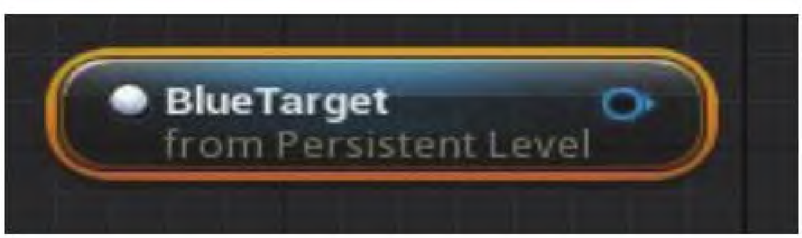



图4-116

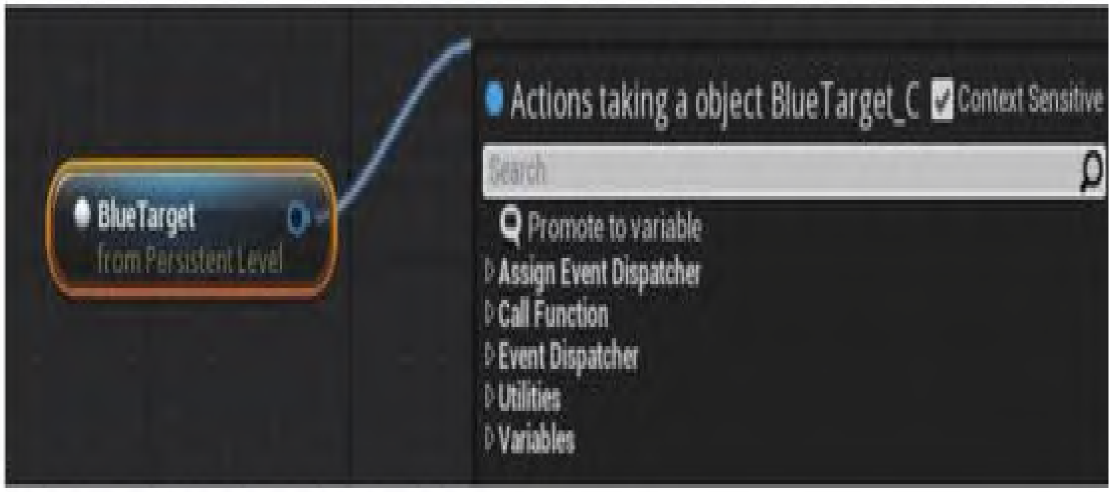



图4-117

**Step 3** 在关联菜单中，导航到Event Dispatcher（事件调度器）>绑定事件到（事件调度器名称）。搜索“Bind（事件名称）”将会快速地弹出正确的选项，如图4-118所示。

图4-118

Call节点将会出现，且Actor引用已经连接到了Target引脚上，如图4-119所示。

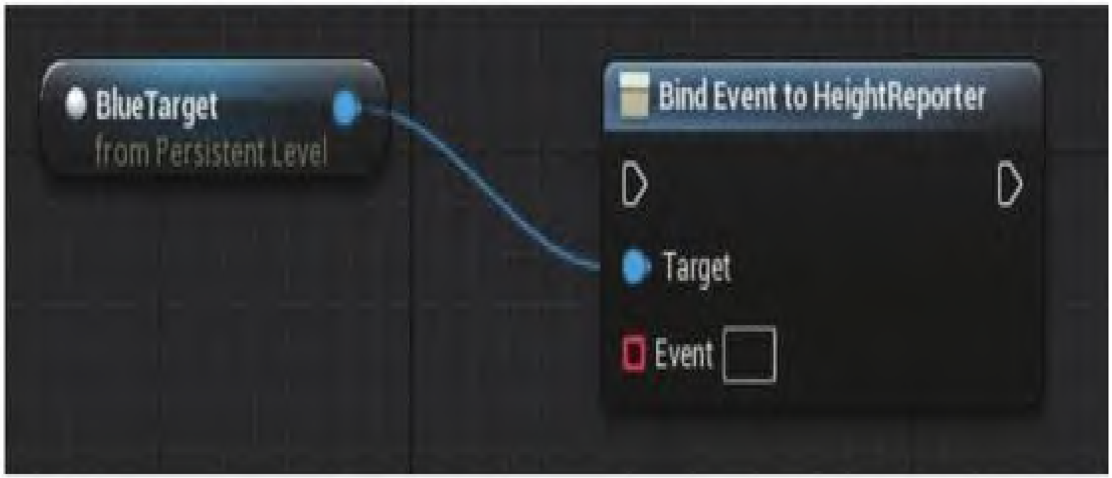



图4-119

## 4.4.2 调用事件调度器

使用Call节点调用一个Event Dispatcher（事件调度器），将会触发绑定到该事件调度器的所有事件。对于每个事件调度器，你可以应用多个Call节点，且你既可以在类蓝图中调用事件调度器也可以在关卡蓝图中进行调用，如图4-120所示。

### 1.在类蓝图中进行调用

**Step 1** 在My Blueprint（我的蓝图）选卡下，从Event Dispatcher（事件调度器）名称处开始拖动鼠标并将其放置到你正处理的图表中。

**Step 2** 在出现的菜单中选择“Call”（调用），如图4-121所示。

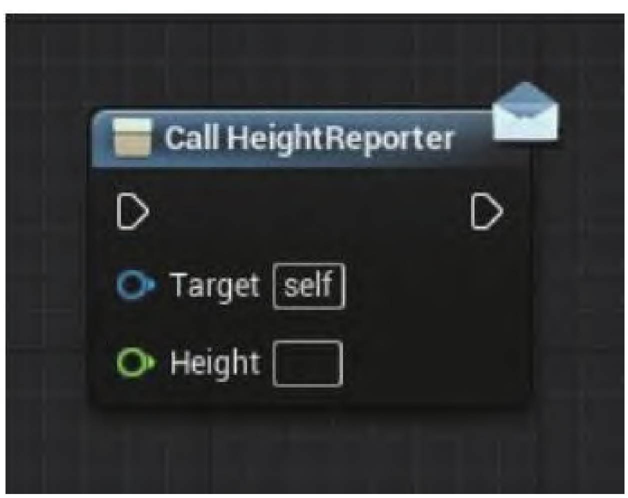


图4-120

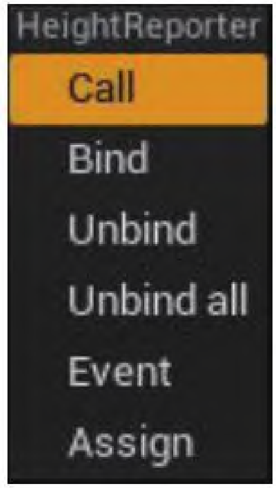



图4-121

另一种方法：

**Step 1** 右击图表。

**Step 2** 在出现的关联菜单中展开Event Dispatcher（事件调度器），如图4-122所示。

**Step 3** 选择Event Dispatcher（事件调度器）下的Call（事件调度器名称），如图4-123所示。

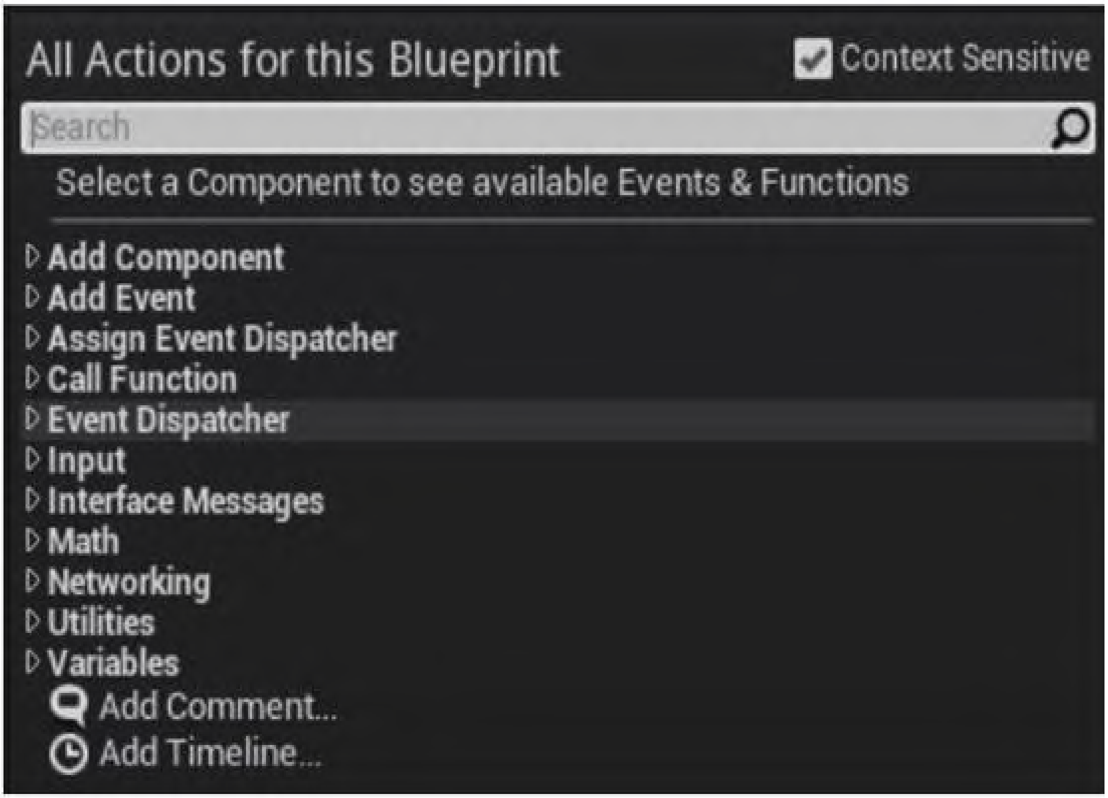

All Actions for this Blueprint
Context Sensitive
Search
Select a Component to see available Events & Functions
Add Component
Add Event
Assign Event Dispatcher
Call Function
Event Dispatcher
Input
Interface Messages
Math
Networking
Utilities
Variables
Add Comment...
Add Timeline...

图4-122

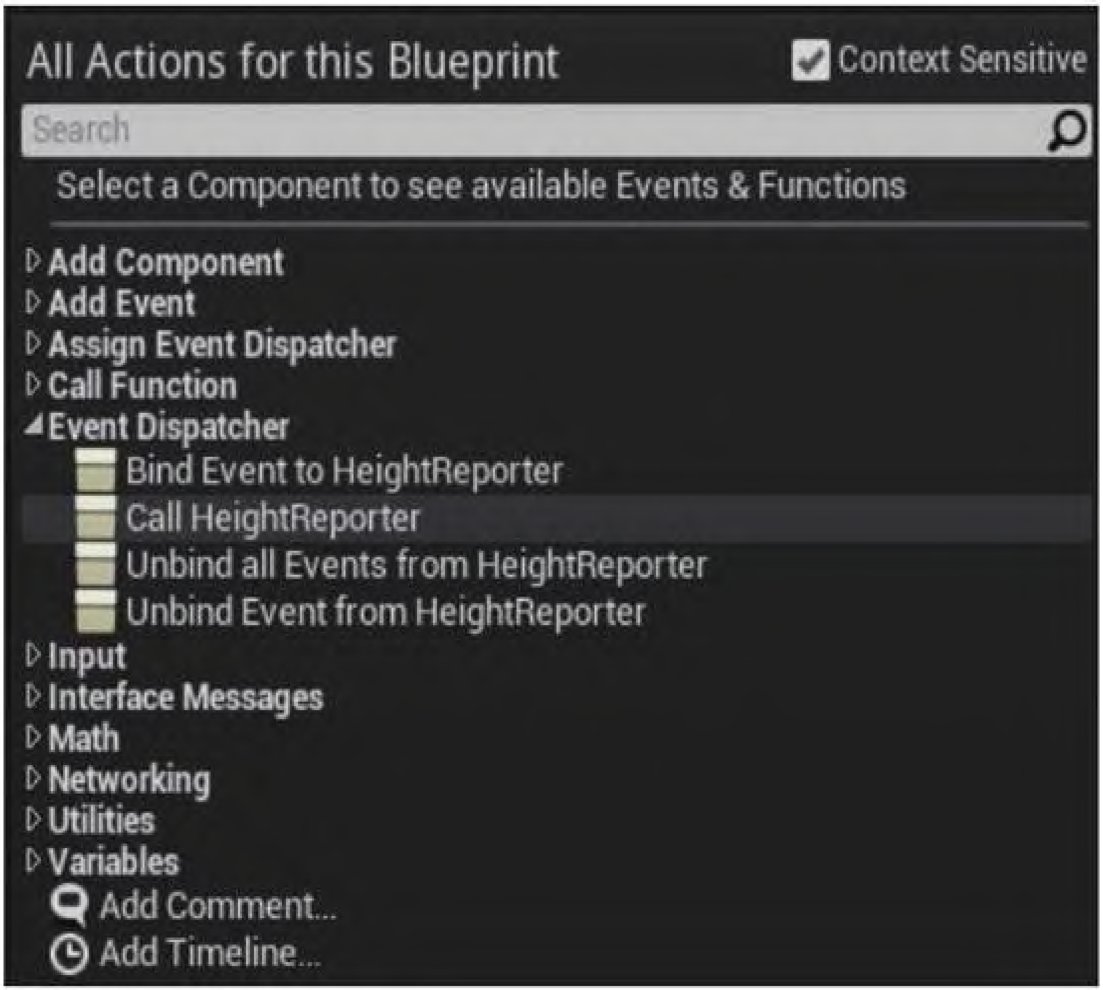



图4-123

### 2. 在关卡蓝图中进行调用

**Step 1**　添加到关卡中你想为其调用事件调度器的Actor的引用，如图4-124所示。

**Step 2**　从该引用节点的输出引脚开始拖动鼠标，然后释放鼠标来显示关联菜单，如图4-125所示。

图4-124

图4-125

**Step 3**　在关联菜单中，导航到Event Dispatcher（事件调度器）>Call（事件调度器名称）。搜索“Event Call”将会快速地弹出正确的选项，如图4-126所示。

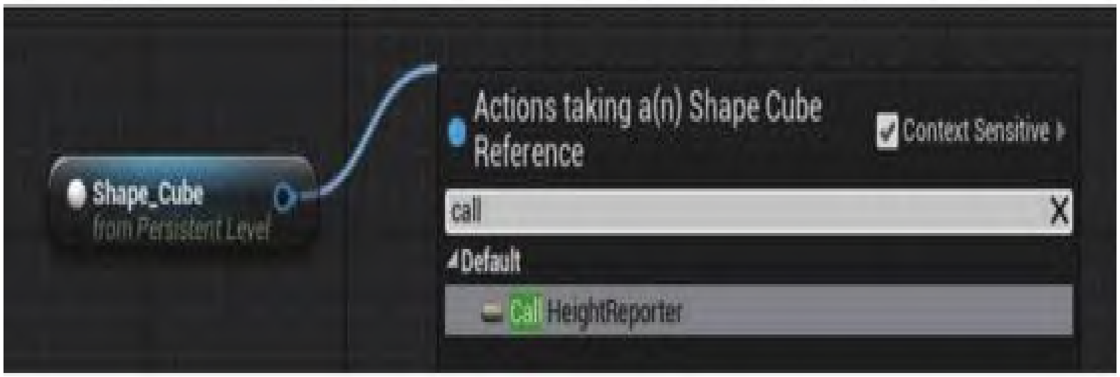



图4-126

Call节点将会出现，且Actor引用已经连接到了Target引脚上，如图4-127所示。

图4-127

### 4.4.3 创建调度事件

通过使用Event Dispatcher（事件调度器）菜单上的Event（事件）选项创建一个具有正确签名的自定义事件，使得该事件可以绑定到该事件调度器上。尽管这个事件节点看上去和没有连接到Bind节点的自定义事件节点类似，但是当执行Call（事件调度器名称）节点时将永远不会触发该事件，如图4-128所示。

图4-128

### 1.创建事件调度器事件节点

事件调度器Event（事件）节点通过将其右上角的红色方框引脚连接到Bind Event（绑定事件）的Event（事件）红色方形输入引脚上，使得该节点和Bind Event（绑定事件）节点相连。所以，如果你已经具有一个针对特定事件调度器的Bind Event（绑定事件）节点，且想为其创建事件节点：

**Step 1** 从Event输入引脚开始拖动鼠标，然后释放鼠标来显示关联菜单。

**Step 2** 从关联菜单中选择Add Custom Event for Dispatcher（为调度器添加自定义事件），如图4-129所示。

**Step 3** 将创建一个事件节点，其该节点已正确地连接到Bind Event（绑定事件）节点上。在空白框中输入你的事件的名称，如图4-130所示。

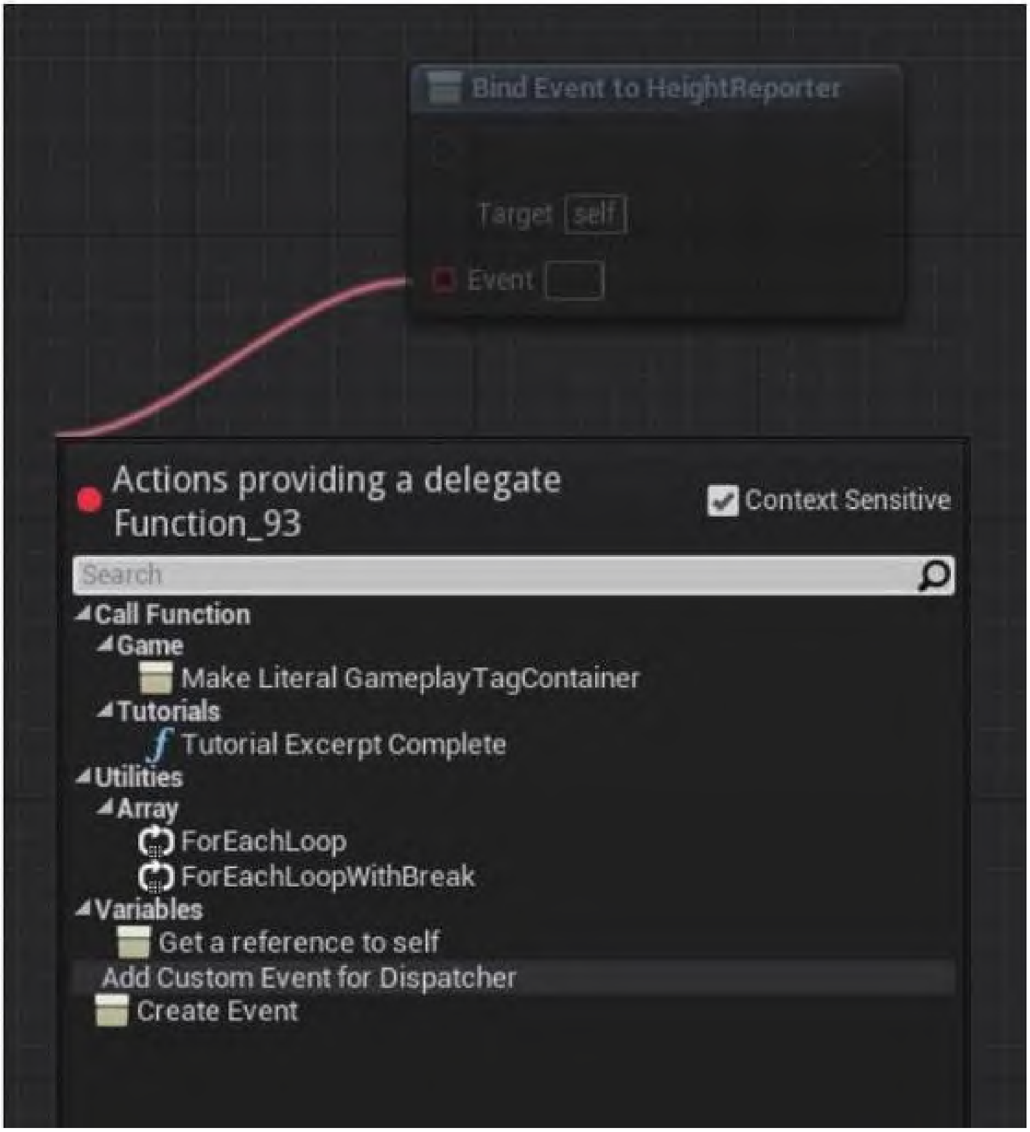

Bind Event to HeightReporter
Target self
Event
Actions providing a delegate Function_93
Context Sensitive
Search
Call Function
Game
Make Literal GameplayTagContainer
Tutorials
Tutorial Excerpt Complete
Utilities
Array
ForEachLoop
ForEachLoopWithBreak
Variables
Get a reference to self
Add Custom Event for Dispatcher
Create Event

图4-129

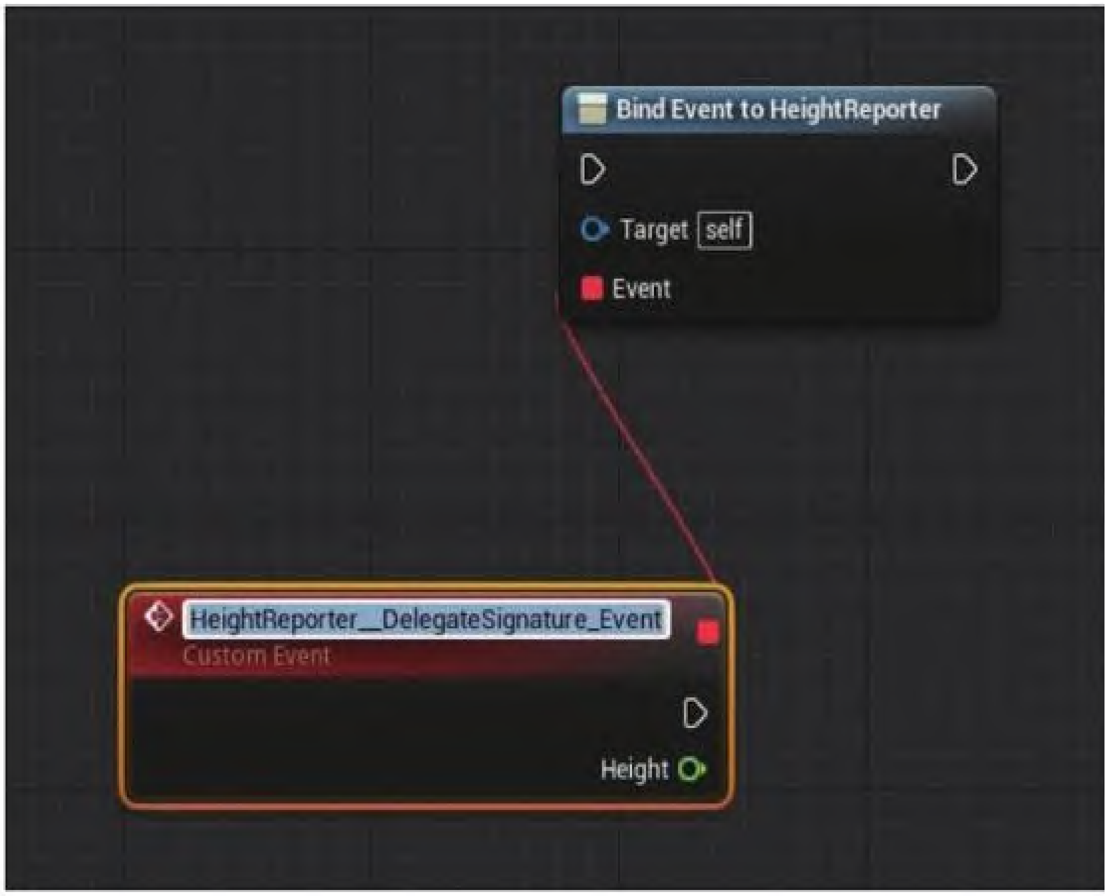



图4-130

从事件调度器菜单或关联菜单中选择Assign（赋值），也会创建一个Bind Event节点和一个对应的Event节点，且两节点已相连。

### 2. 在类蓝图中进行创建

**Step 1** 在My Blueprint（我的蓝图）选卡下，从Event Dispatcher（事件调度器）名称处开始拖动鼠标并将其放置到你正处理的图表中。

**Step 2** 在出现的菜单中选择Event（事件），如图4-131所示。

**Step 3** 输入你的事件的名称，如图4-132所示。

图4-131

图4-132

### 3. 在关卡蓝图中进行创建

在关卡蓝图中，可以创建一种特殊类型的Event Dispatcher（事件调度器）事件，此时事件会自动绑定到事件调度器上。创建这些事件的步骤和创建默认事件（比如OnClicked或OnOverlap事件）一样。关卡蓝图文档中提供了关于这个过程的详细介绍。

这些特殊事件是独特的，在游戏开始时会进行自动绑定。因此，任何时候执行Unbind All（解除所有绑定）节点都将解除绑定这些事件。而通过把这些事件的代理引脚和游戏中其他地方执行的Bind Event节点相连，还可以重新绑定它们。

4.5

# 蓝图接口

## 4.5.1 概述

Blueprint Interface（蓝图接口）是一个函数或多个函数的集合，仅有函数名称，没有函数实现，该接口可以添加到其他蓝图中。任何具有该接口的蓝图，都一定会具备这些函数。在每个添加了接口的蓝图中，都可以提供接口的这些函数的功能。从本质上讲，这和一般编程中的接口的概念一样，允许通过一个公共接口来共享及访问多种不同类型的对象。简单地讲，Blueprint Interfaces（蓝图接口）允许不同的Blueprints（蓝图）间彼此共享及发送数据。

蓝图接口的创建和其他蓝图类似，可以通过编辑器中的内容创建器进行创建，但它们有一些局限性，因为它们不能：

•添加新变量。

•编辑图表。

•添加组件。

使用蓝图接口，允许你通过一种通用方法，来和具有某些特定功能的多种不同类型的对象进行交互。这意味着你可以具有属于完全不同类型但又共有一些特性的对象，比如一辆汽车和一棵树，它们都可以受到武器的射击，并且都会受到伤害。通过创建一个包含OnTakeWeaponFire函数的蓝图接口，并让汽车和树都实现那个蓝图接口，当汽车或树两者中任何一个被射中时，你可以把它们看成相同的类型，仅需简单地调用OnTakeWeaponFire函数即可。关于如何实现蓝图接口，请参照实现蓝图接口页面。

## 4.5.2 创建蓝图接口

在内容浏览器中创建蓝图接口非常简单，仅需右击并从关联菜单中选择Blueprint Interface（蓝图接口），然后命名你的新接口即可。

一旦创建了蓝图接口，你就可以编辑该接口中的函数。

### 4.5.3 编辑蓝图接口

蓝图是在一个特殊版本的蓝图编辑器中进行编辑的。因为你不能创建你自己的变量、图表或组件，所以编辑接口的过程和编辑标准蓝图的过程有很大差别。

当你第一次打开一个没有函数的接口时，编辑器，如图4-133所示。

图4-133

注意，没有图表视图。这是因为此时图表还没有函数。

### 1. 添加函数

函数是接口的主要组成部分。接口函数没有实现。它们的存在仅是简单地定义输入和输出。这些函数可以用于通过接口发送数据，或者可以被任何实现那个接口的蓝图所重写。

要想添加一个新函数：

**Step 1**　单击 按钮。

**Step 2**　在我的蓝图面板中，输入该新函数的名称，如图4-134所示。

**Step 3**　新的图表区域将出现，并具有该新函数。注意，该函数既没有输入也没有输出，如图4-135所示。

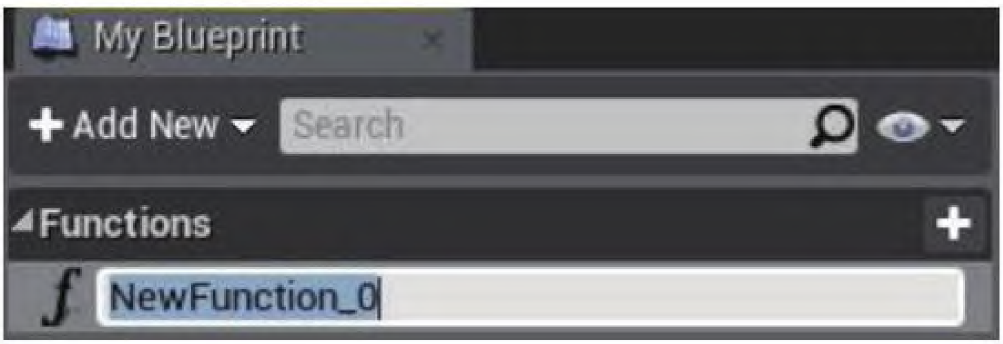



图4-134

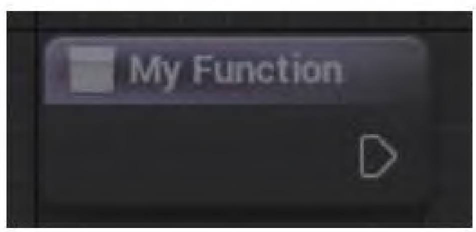



图4-135

### 2. 编辑函数签名

由于接口函数没有实现，作为设计师，你所能做的就是指定一系列的具有特定类型的输入和输出。

要想编辑函数的签名：

**Step 1**　在详细信息选卡中，滚动到Inputs（输入类目）并单击 New 按钮来创建一个新的输入参数，如图4-136所示。

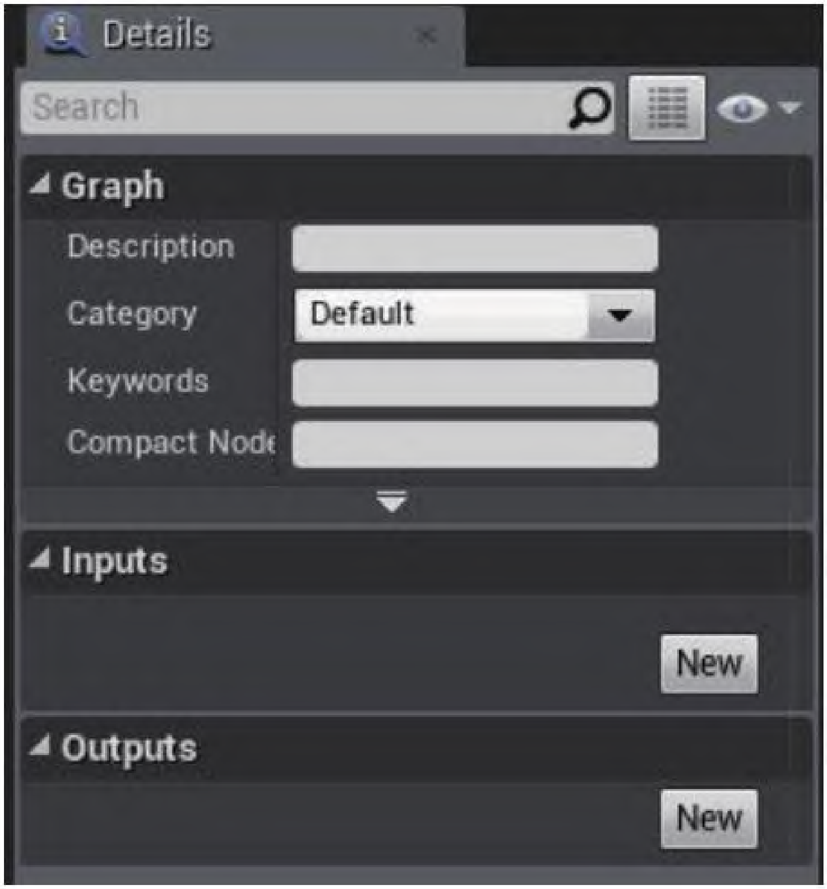



图4-136

**Step 2**　根据需要，设置Input Name（输入名称）和Type（类型）。你也可以通过名称旁边的 ▷ 按钮展开该输入，从设置一个Default Value（默认值），如图4-137和图4-138所示。

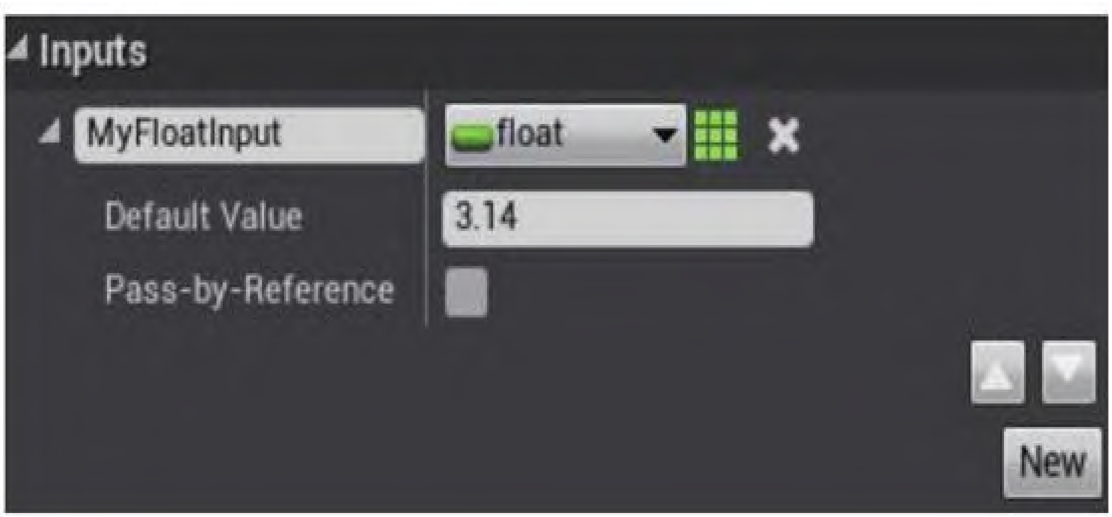


图4-137

图4-138

**Step 3** 添加输出的方式也是一样的。注意观察图表是如何自动更新来显示所添加的输出的，如图4-139和图4-140所示。

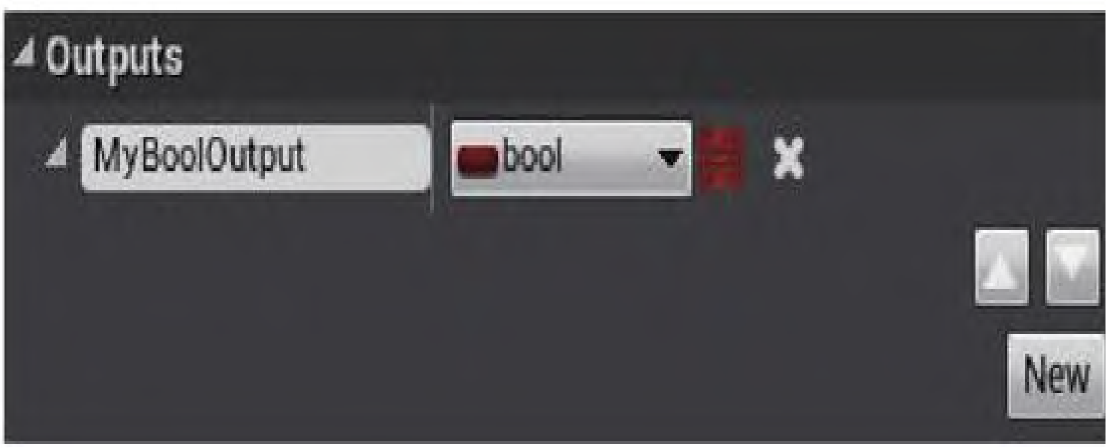



图4-139

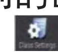
图4-140

### 3. 其他注意事项

•在任何输入或输出参数上，你可以单击 按钮来删除那个参数。

•输入参数可以通过参数名称文本框下面的小文本框设置一个默认值，但是你必须在详细信息面板中展开该属性项来查看它。

•当参数属性处于展开状态时，你可以使用详细信息选卡中的 和 按钮来修改出现在节点上的输入参数和输出参数。

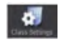
•对于任何包含了需要通过服务器进行复制的函数的接口，你可以为其应用Replicates（复制）复选框。通过首先单击  按钮，可以在详细信息选卡中找到该复选框。

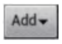
## 4.5.4　实现蓝图接口

使用蓝图接口，允许你通过一种通用方法，来和具有某些特定功能的多种不同类型的对象进行交互。这意味着你可以具有属于完全不同类型但又共有一些特性的对象，比如一辆汽车和一棵树，它们都可以受到武器的射击，并且都会受到伤害。通过创建一个包含OnTakeWeaponFire函数的蓝图接口，可以让汽车和树都实现那个蓝图接口，当汽车或树两者中任何一个被射中时，你可以把它们看成相同的类型，仅需简单地调用OnTakeWeaponFire函数即可。

### 1. 添加接口

在原生代码（C++）中定义的蓝图接口及接口，都可以添加到除了关卡蓝图之外的任何类型的蓝图中。

Step 1　单击工具条上的  按钮。

Step 2　在详细信息选卡中，滚动到Interfaces（接口）类目处。单击  按钮来查看可用的接口列表。单击你想使用的接口来添加它，如图4-141和图4-142所示。

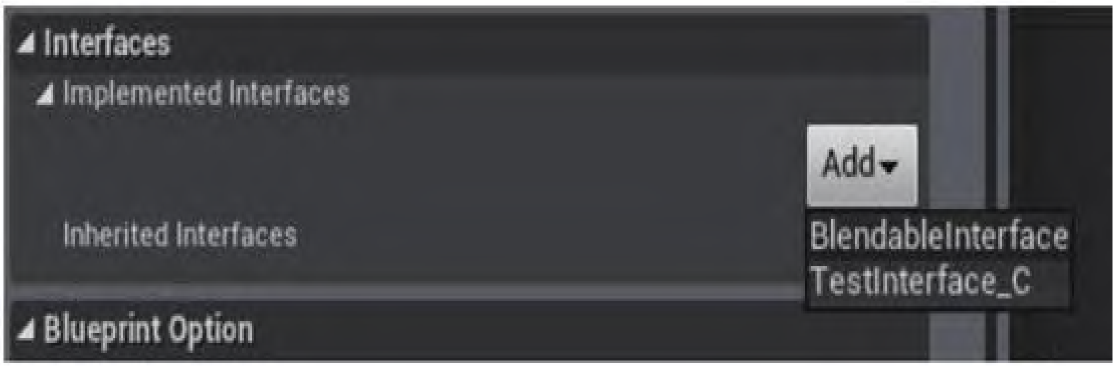



图4-141

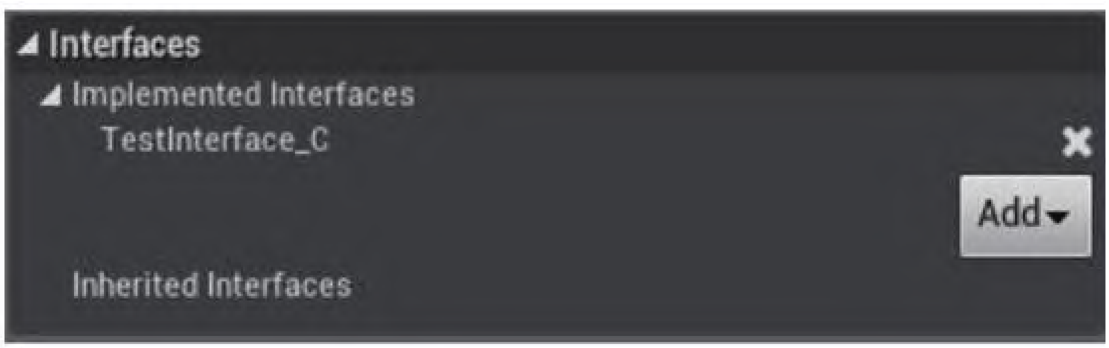



图4-142

通过单击Implemented Interfaces（已实现的接口）列表中每个接口旁边的 按钮，可以删除接口。你也可以将蓝图接口迁移为正常的蓝图函数。关于更多信息，请参照蓝图接口迁移。

接口类目中的Inherited Interfaces（继承的接口）部分会自动地填充该对象所继承的任何接口（也就是，在当前蓝图继承的蓝图中实现的接口）。

### 2. 实现接口成员

当把一个接口添加到一个蓝图中时，那个蓝图必须实现该接口的函数，才能在调用函数时执行动作。这可以通过在事件图表中为接口函数放置一个

Event（事件）节点来完成。连接到该Event（事件）上的节点定义了函数的功能或实现。

**Step 1** 右击该蓝图的事件图表，并选择Add Event>Event Example Function（把你正在实现的接口中的函数名称替换为Example Function）。Event（事件）节点将显示在图表选卡中，如图4-143和图4-144所示。

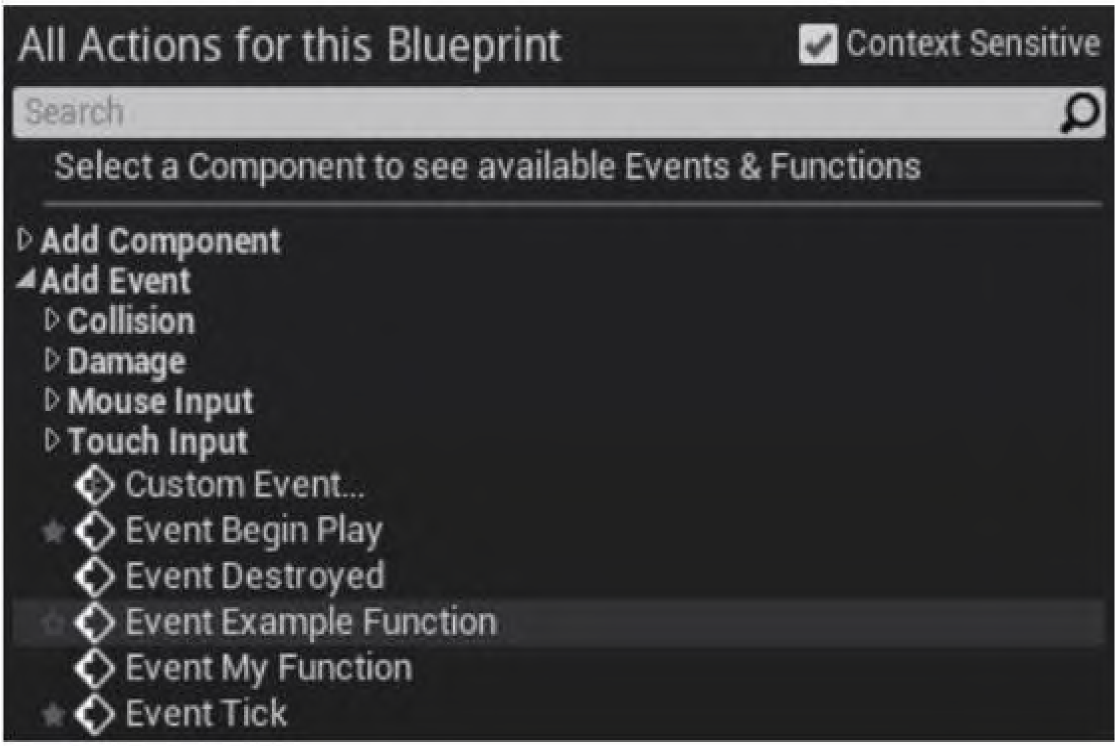



图4-143

图4-144

**Step 2** 创建连接到该事件的执行引脚的节点网络，来为其设置功能，如图4-145所示。

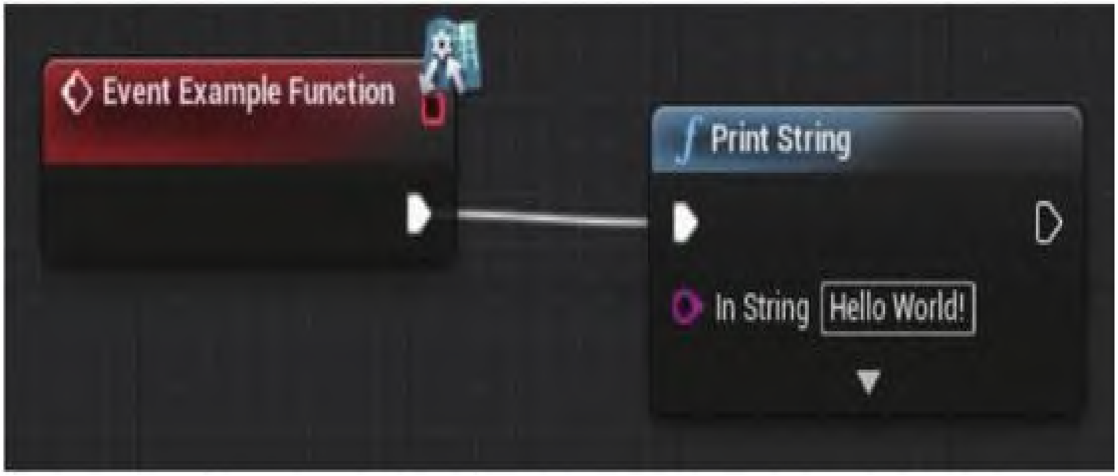



图4-145

### 3. 调用接口函数

接口的函数通过使用Interface Message（接口消息）节点来执行，这些节点除了在目标没有实现该接口时会执行失败且没有征兆提示外，其外观和行为和正常的函数调用一样，如图4-146所示。

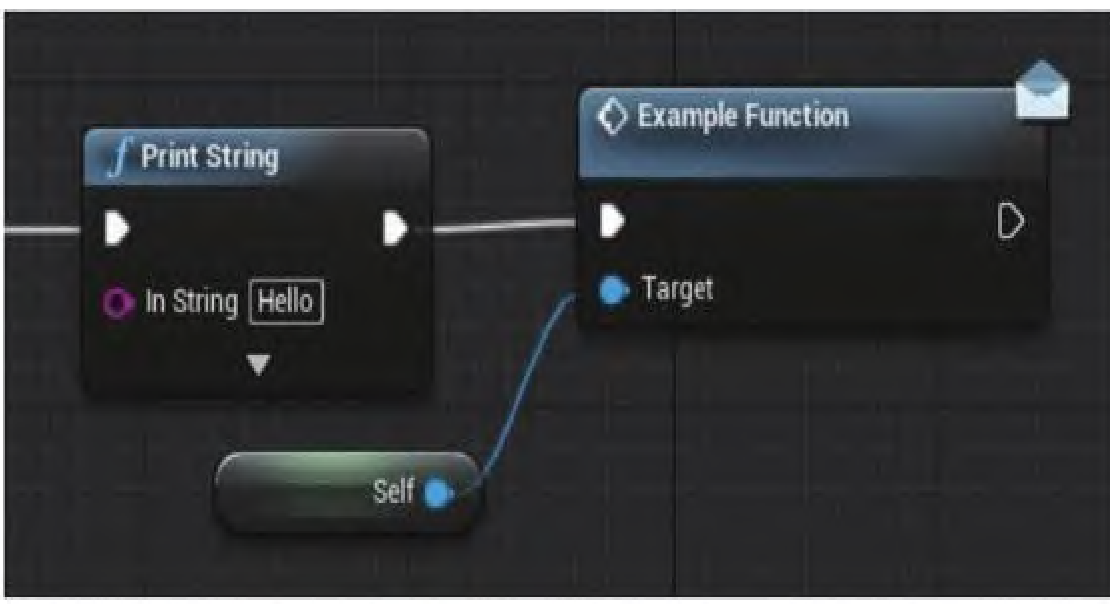



图4-146

### 4. 在其他蓝图上调用接口函数

接口在不同蓝图间进行通信方面是非常有用的。你可以在其他蓝图上调用一个接口中的函数。当满足以下两个条件时就可以这样做：

•两个蓝图实现了具有所需函数的同一个接口。

•调用接口的蓝图知道另一个蓝图在游戏中的实例的名称。

当在事件图表中创建一个Interface Function（接口函数）节点时，你通常看到关联菜单中列出了该函数的两个版本，如图4-147所示。

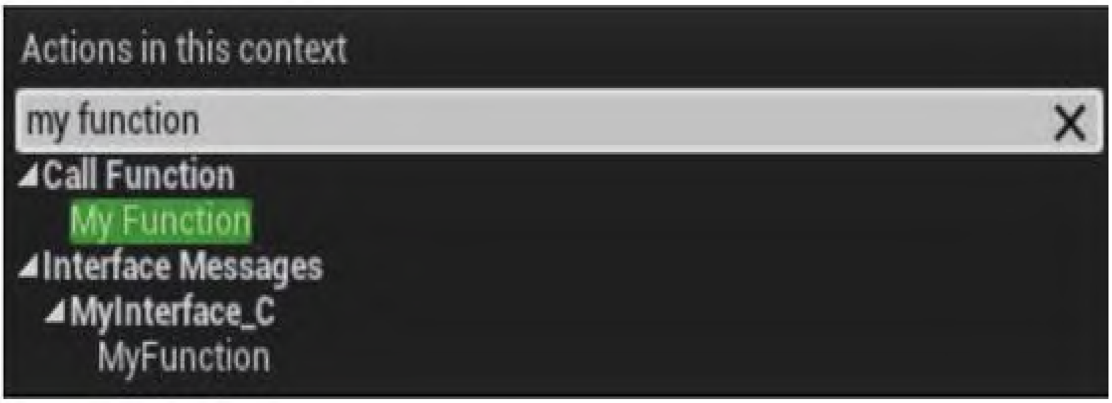



图4-147

值得注意的是列在Call Function（调用函数）下面的版本是本地版本，旨在仅从这个蓝图中调用该函数。

在Interface Messages（接口消息）下面列出的版本的设计目的是：在其他实现了同样接口的蓝图中上通过该接口调用该函数。

在节点标题的下面有些标签，以帮助你进行区分：

| | |
|---|---|
| | |
| 本地版本 | 接口消息版本 |

调用一个Interface Message Function（接口消息函数），需要你提供你在其上面调用该函数的适当蓝图的实例。

注意，在这个图片中，我们正在发送另一个蓝图作为Target（目标）输入。这将会在提供的蓝图上调用该函数。如果那个蓝图没有实现该接口，那么将不会发生任何事情，如图4-148所示。

要想指定一个关卡中存在的蓝图，最好创建一个Actor类型的公有变量，并使用关卡编辑器的详细信息选卡在那个变量的值处赋予适当的蓝图。

### 5.重写接口函数

接口函数的其中一个最重要的方面是：因为它们没有自己的实现，所以每个实现该接口的蓝图可以独立地定义功能。

一旦蓝图实现了一个接口后，该接口的函数就会显示在我的蓝图面板中，如图4-149所示。

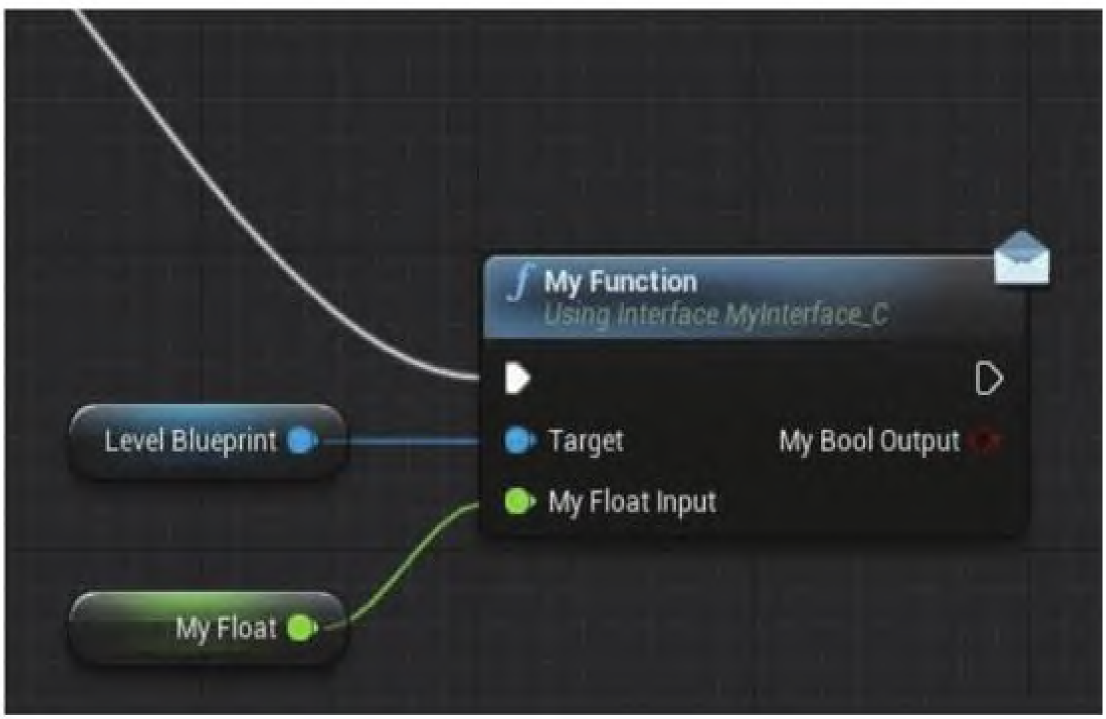



图4-148

图4-149

如果你双击那个函数，将会在蓝图编辑器中打开它。这显示了该函数的本地实现。这里所定义的任何功能对于这个蓝图都是唯一的。通过这种方式，你可以为实现该接口的每个蓝图定义定制的功能。

比如，在这个图片中，我们看到一个在蓝图中打开的基本接口函数。目前，函数没有任何功能，如图4-150所示。

图4-150

我们可以创建一个简单的节点网络，来进行数值比较、输出值等类似下面的处理，如图4-151所示。

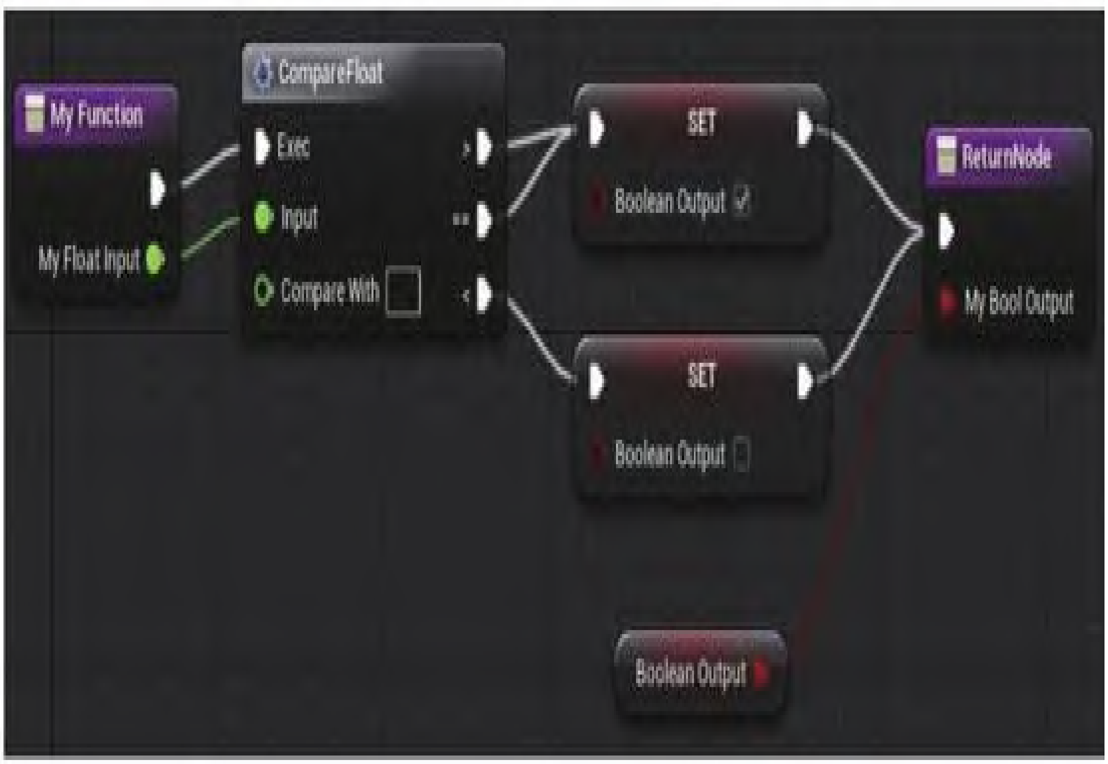



图4-151

现在，当从本地调用函数时，函数将取入一个浮点值，将其和数值10进行比较，然后向屏幕输出布尔型结果，如图4-152所示。

Event Begin Play
My Function
Target is TestBP
Target
self
My Bool Output
My Float
My Float Input
Print String
In String

图4-152

第5章

# 蓝图编辑器

5.1

## 蓝图编辑器模式

Class Blueprint Editor（类蓝图编辑器）的每种模式呈现不同版本的工具栏和菜单。

要想切换到另一种模式，请单击工具栏的相应按钮，如图5-1所示。

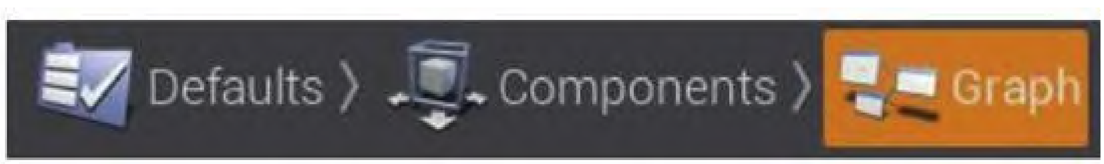



图5-1

Defaults Mode ：

Blueprint Defaults（蓝图默认值）或Defaults（默认）模式显示了蓝图的所有属性，包括蓝图内包含的变量、原生代码中定义的组件、以及复制、导航和碰撞属性。当创建蓝图的新实例时将会使用在Defaults（默认）模式中对这些属性设置的值，如图5-2所示。

蓝图编辑器组件模式：

在Components Mode（组件模式）中，你可以把组件添加到你的关卡中，管理现有组件的详细信息及排列组件。组件模式中的组件列表是以层次化结构呈现的，如图5-3所示。

Graph Mode：

图形模式，如图5-4所示。

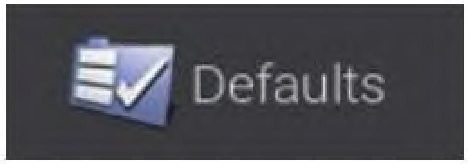



图5-2

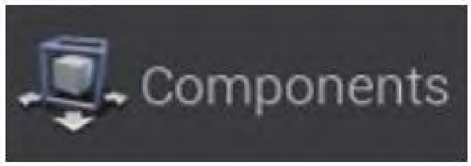



图5-3

图5-4

Blueprint Defaults（蓝图编辑器默认值）包含了关于蓝图的默认设置、属性及蓝图中包含的所有变量的信息。你可以修改这些设置来满足你自己的需要，如图5-5所示。

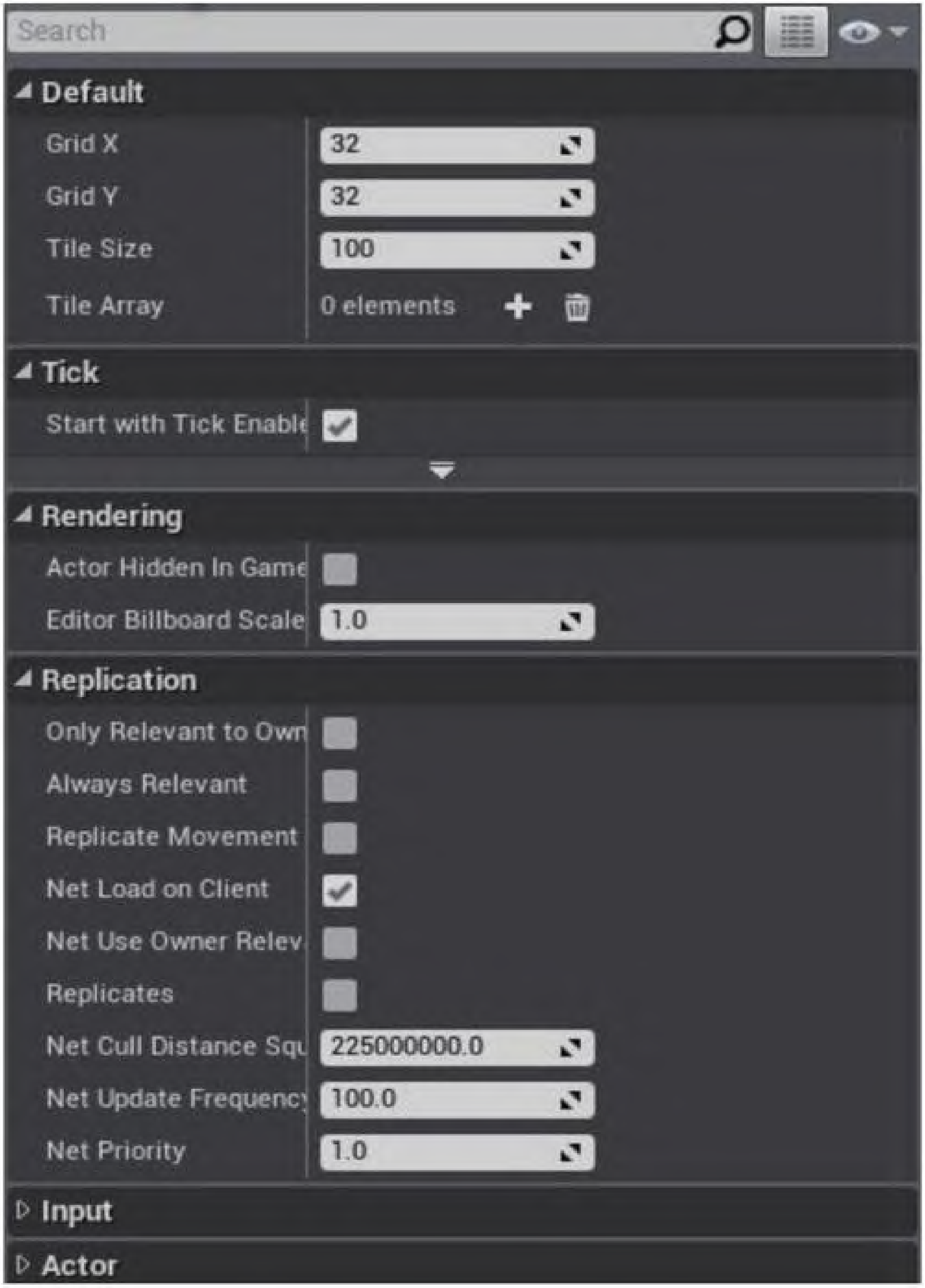



图5-5

蓝图默认值选卡包含以下部分：

| 项　目 | 描　述 |
|---|---|
| Default（默认） | 如果你在创建某个变量时没有指定除 Default 之外的其他类目，那么这里则列出所有变量及它们的值。注意你必须编译该蓝图，才能在 Blueprint Defaults（蓝图默认值）选卡中显示它的变量 |
| Rendering（渲染） | 包含了如何在游戏中渲染基于蓝图的 Actor 的相关属性的信息 |
| Replication（复制） | 包含了指示你的 Actor 如何在网络游戏中表现的属性相关的信息，比如它的网络优先级及同其他客户端的关联 |
| Input（输入） | 包含了基于该蓝图的 Actor 如何对输入做出反应的相关信息 |
| Actor | 包含了基于这个蓝图的 Actor 相关的信息，比如它们是否会受到伤害。注意，对于关卡蓝图来说，在这部分中仅 Tags（标签）属性是有用的 |

5.2

## 蓝图类UI

蓝图类UI，如图5-6所示。

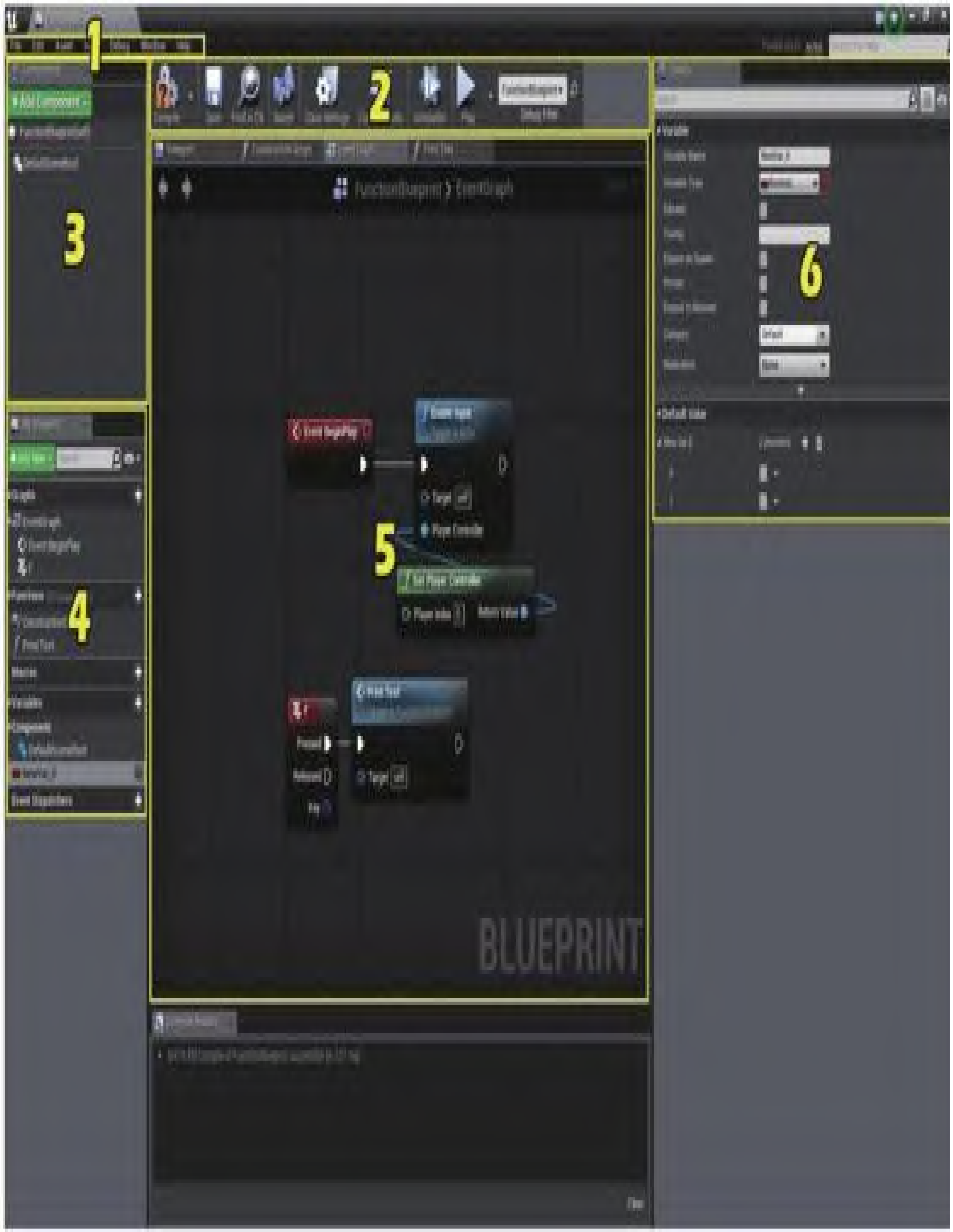



图5-6

Blueprint Class（蓝图类）编辑器模式默认包含以下几个选卡。

| 默认可见的用户界面组件 | 窗口菜单中可用的组件 |
| --- | --- |
| 菜单 | 调试 |
| 工具栏 | 类默认值 |
| 组件 | 编译器运算结果 |
| 我的蓝图 | 搜寻结果 |
| 图表编辑器 | 视口 |

5.3

## 仅包含数据的蓝图UI

Data-Only Blueprint（仅包含数据的蓝图）是指仅包含代码（以节点图表的形式）、变量及从其父类继承的组件的类蓝图。仅包含数据的蓝图允许你调整及修改继承的属性，但是不能添加新元素。从本质上讲，这些蓝图是原型的替代物，设计人员可以使用它们来调整属性或者设置具有变种的项目。

Data-Only Blueprints（仅包含数据的蓝图）是在合并的属性编辑器中进行编辑的，但是也可以通过使用完整的蓝图编辑器来添加代码、变量或组件，来将其转换为完整的蓝图。

5.4

# 蓝图接口用户界面

## 5.4.1 概述

Blueprint Interface（蓝图接口）是一个函数或多个函数的集合，仅有函数名称，没有函数实现，该接口可以添加到其他蓝图中。任何具有该接口的蓝图，都一定会具备这些函数。在每个添加了接口的蓝图中，都可以提供接口的这些函数的功能。从本质上讲，这和一般编程中的接口的概念一样，允许通过一个公共接口来共享及访问多种不同类型的对象。简单地讲，Blueprint Interfaces（蓝图接口）允许不同的Blueprints（蓝图）间彼此共享及发送数据。

蓝图接口的创建和其他蓝图类似，可以通过编辑器中的内容创建器进行创建，但它们有一些局限性，因为它们不能：

•添加新变量；

•编辑图表；

•添加组件。

## 5.4.2 蓝图接口编辑器用户界面

当你第一次打开一个蓝图接口时，它的用户界面看上去是非常简单的，仅包含以下内容，见图5-7和下表。

| 默认可见的用户界面组件 | 窗口菜单中可用的组件 |
|---|---|
| 菜单 | 调试 |
| 工具栏 | 编译器结果 |
| 我的蓝图 | 查找结果 |
| 详细信息 | |

然而，如果你执行以下任何操作，此用户界面将会有少许变化：

在我的蓝图面板中通过双击一个现有函数来打开它。

通过在我的蓝图面板中右击一个新函数并从关联菜单的Add New（添加新项）部分选择Function（函数）。

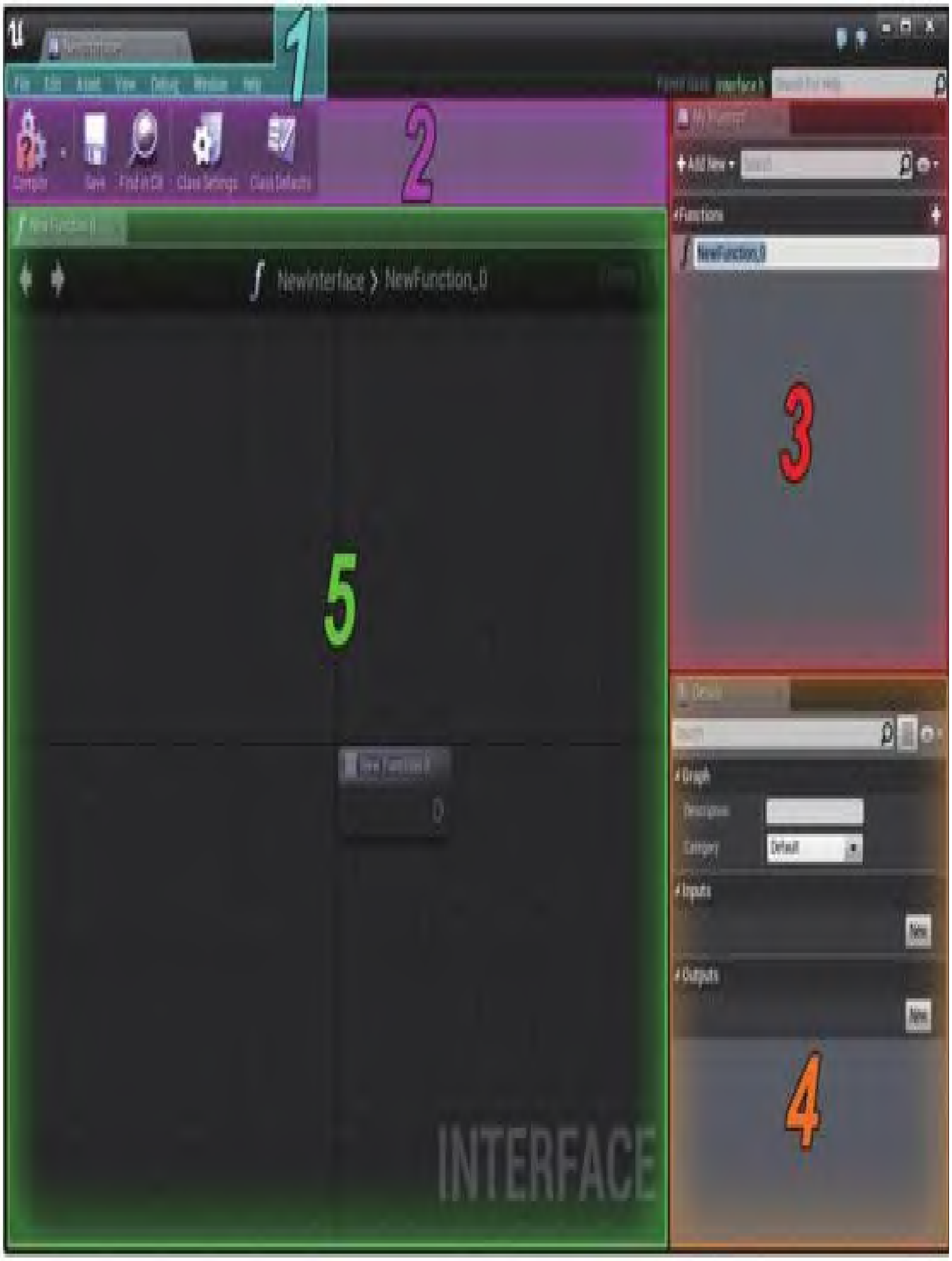



图5-7

你一执行这些操作后，该图表将呈现为如图5-8所示。

1
2
3
4
5
Compile
Save
Blueprint Props
My Blueprint
Function
SetObjective
ToggleMenu
GetPlayerHealth
ToggleMenu
SetObjective
BPI_HUD_Interface_Events > SetObjective
Graph
Inputs
Objective
ObjectiveName
Outputs
INTERFACE

图5-8

5.5

# 关卡蓝图UI

## 5.5.1　概述

关卡蓝图是一种特殊类型的蓝图，是作用于整个关卡的全局事件图表。关卡事件或者关卡中的Actor的特定实例，用于激活以函数调用或者流程控制操作的形式呈现的动作序列。熟悉UE3的人应该非常熟悉这个概念，因为这和UE3中的Kismet的工作原理非常类似。

关卡蓝图提供了针对关卡动态载入、Matinee以及给放置到关卡中的Actor绑定事件的控制机制。

## 5.5.2　接口

在编辑关卡蓝图时，蓝图编辑器包含以下部分，见图5-9和下表。

1
2
3
4
5
Compile
Search
Class Settings
Play
Example_Map › EventGraph
Right-Click to Create New Nodes.
LEVEL BLUEPRINT


图5-9

| 默认可视 UI 组件 | 窗口菜单中可用的组件 |
|---|---|
| 菜单 | 类默认值 |
| 工具条 | 编译器运算结果 |
| 我的蓝图 | 调试 |
| 详细信息 | 搜索运算结果 |
| 图表编辑器 | 面板 |
| | 视口 |

5.6

## 宏库UI

Blueprint Macro Library（蓝图宏库）是一个存放了一组宏的容器，或者是可以像节点那样放置到其他蓝图中的自包含图表。这些蓝图宏库非常节约时间，因为它们存放了常用的节点序列及针对执行和数据变换的输入和输出。

宏会在所有引用它们的图表间共享，但是如果它们在编译过程中是合并的节点，那么它们会自动地展开为图表。这意味着蓝图宏库不需要进行编译，但是对宏所做的修改，仅当重新编译了包含引用该宏的图表的蓝图时，这些修改才会反映在图表中。

和蓝图界面相同，打开宏库的蓝图编辑器后可看到一个不带图表的简化UI，见图5-10和下表。

1
2
3
4
5
NewMacroLibrary › NewMacro_0
Drag Off Pins to Create/Connect New Nodes.
Inputs
Outputs
MACRO
Macros
NewMacro_0
Graph
Default


图5-10

| 默认可见 UI 组件 | 窗口菜单中可选 |
| --- | --- |
| Menu | Debug |
| Toolbar | Find Results |
| My Blueprint | Palette |
| Details | |
| Graph Editor | |

5.7

# 编译器结果

## 5.7.1　概述

编译器结果面板提供了编译你的蓝图脚本的反馈，告知你蓝图脚本编译是否成功或者是否有错误或警告。任何时候当发现错误或警告时，编译器结果面板会提供关于错误的信息并给出超链接，使你可以直接跳转到图表视图中存在问题的根源处。

## 5.7.2　界面

编译器结果的用户界面非常简单，主要由编译成功、编译警告及编译错误（失败）信息的自动填充列表构成。在任何可行的地方，你可以将鼠标悬停到每条信息上获得更多详细信息，并且所有警告或错误都会提供一个超链接，使你可以直接跳转到该信息的源头处，如图5-11所示。

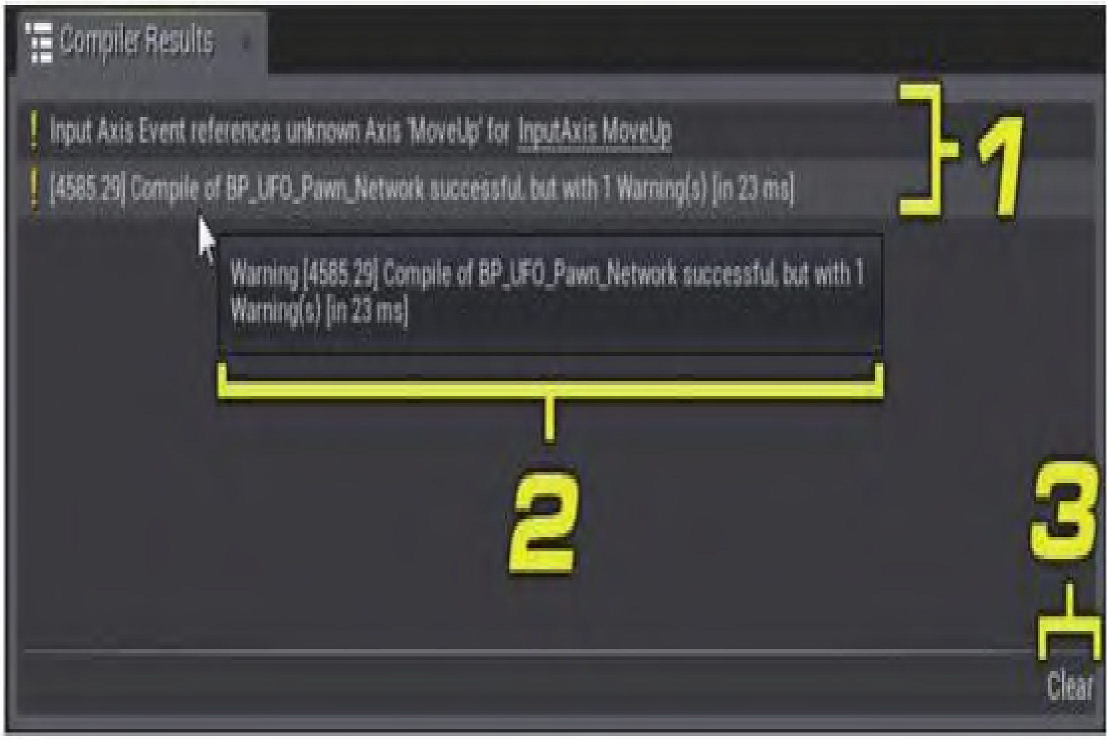



图5-11

•Result List（结果列表）：这是个自填充的列表，显示了你的编译结果。将会呈现到任何错误及警告的超链接。如果编译成功，你还会看到总的编译时间，以毫秒为单位。

•Pop-up Help（弹出式帮助）：通过把鼠标悬停到列表中的每一项上，你可以获得关于该项本质的更多信息。如果有超链接，你可以直接跳转到出问题的节点。

•Clear Button（清除按钮）：该按钮简单地清除现有的结果列表。

## 5.7.3 打开编译器结果面板

默认情况下，编译结果面板并不总是显示，但是可以在蓝图编辑器的窗口菜单中找到它。当蓝图编译产生了错误或警告时，也会自动打开该面板。一般，该面板将会出现在当前图表面板的底部。

## 5.7.4 错误和警告浏览

任何时候，当你的编译过程中产生警告或错误时，编译器结果面板可以使你执行以下操作。

•将鼠标悬停到一条信息上来查看详细信息，如图5-12所示。

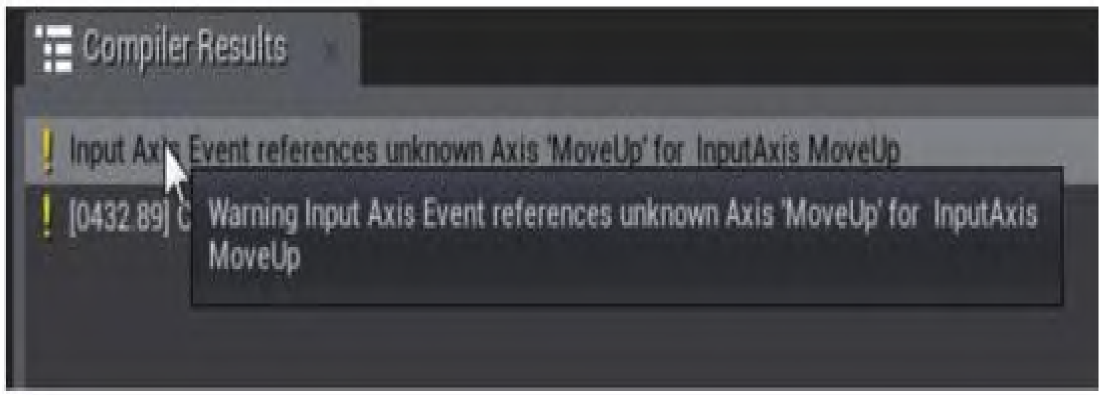



图5-12

•将鼠标悬停到一条信息尾部的超链接上，可以查看出问题的节点的链接，如图5-13所示。

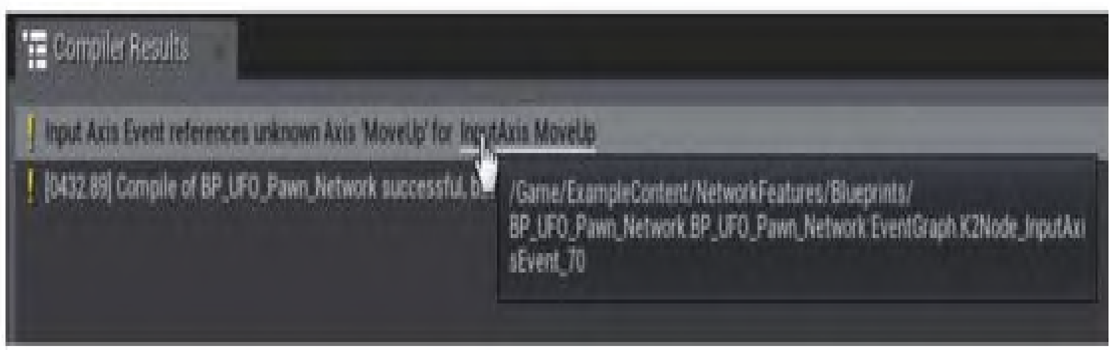



图5-13

•单击一条信息来直接跳转到图表视图中出问题的节点或者可视化脚本的某个部分，如图5-14所示。

图5-14

•单击面板右下角的清除按钮来清除现有的所有信息，如图5-15所示。

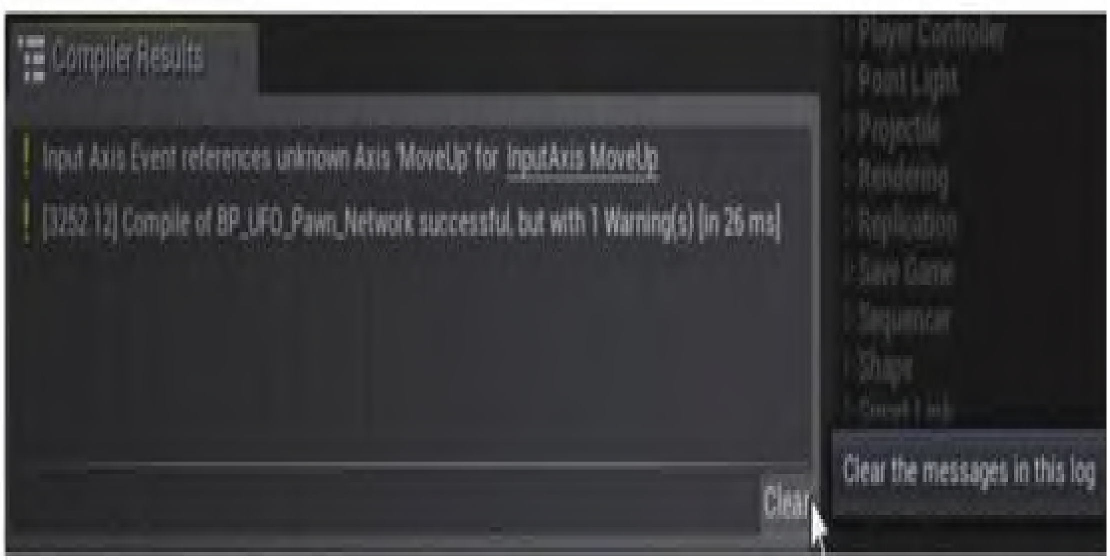


图5-15

5.8

# 调试面板

## 5.8.1　概述

调试面板提供了目前可以用于调试蓝图的一系列调试工具，比如Breakpoints（断点）及Watch Values（查看值）。当处于Play In Editor（在编辑器中运行）时，它还允许你访问Execution Trace（执行追踪），向你展示了在给定蓝图中发生的每个节点的执行情况。

## 5.8.2　界面

调试面板的界面根据你是在编辑器中运行游戏还是在编辑器中模拟游戏的不同而有所变化。

当没有处于运行或模拟过程中时，调试面板的界面如图5-16所示。

图5-16

当没有运行游戏时，调试面板列出了你的当前蓝图中的所有查看值和断点。

当处于运行或模拟过程中时，调试面板界面如图5-17所示。

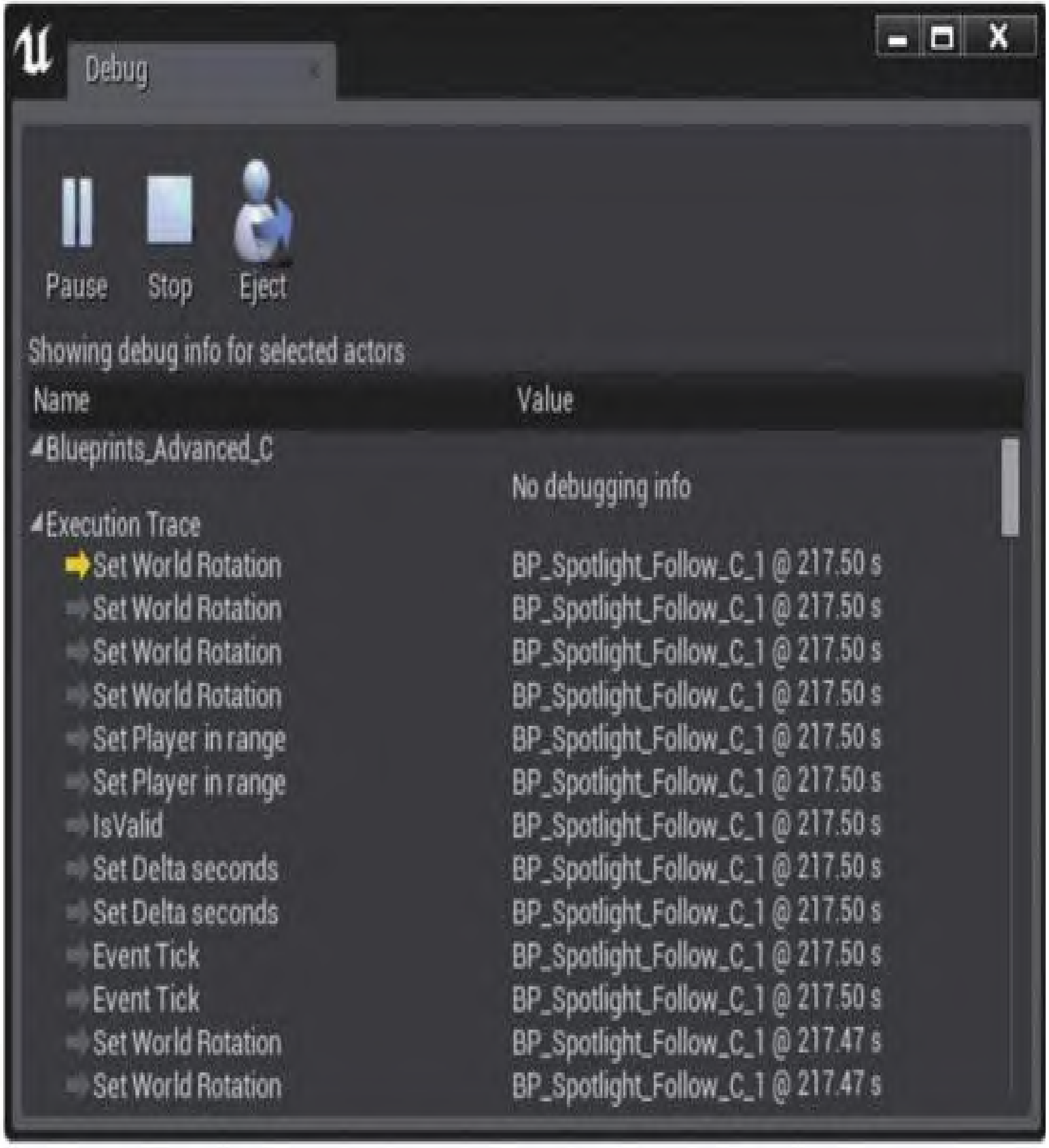



图5-17

当处于蓝图运行或模拟过程中时，调试面板显示了调试信息及Execution Trace（执行追踪），Execution Trace显示了执行每个节点所花费的时间。

5.9

# 详细信息面板

## 5.9.1　概述

Details（详细信息）面板是一个情境关联的区域，使得可以在蓝图编辑器中编辑选中项的属性。它包含一个用于快速访问特定属性的搜索条，并且一般还会包含一个或多个可合并的类目，用于组织其中所包含的属性。

详细信息面板也是你处理很多蓝图编辑工作的地方，包括：

•编辑蓝图变量的过程，包括修改名称、类型、及该变量是否是一个数组。

•单击Blueprint Props（蓝图属性）按钮后，实现蓝图接口。

•为蓝图函数添加输入和输出。

•为选中的组件添加事件。

## 5.9.2　界面

界面如图5-18所示。

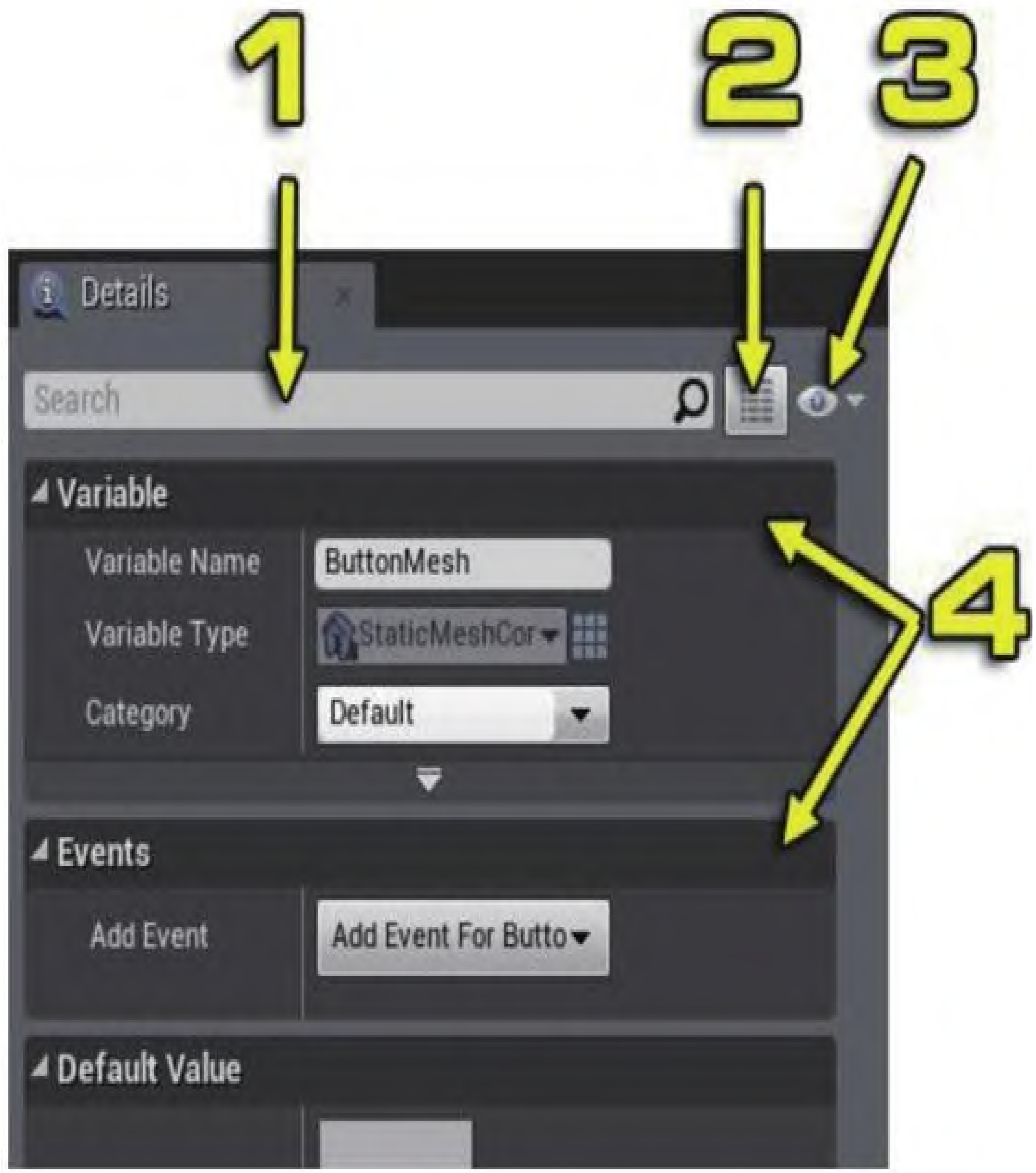



图5-18

•Search Filter（搜索过滤器）：输入你需要的属性的名称，这些属性就会显示在过滤器的下方。

•Property Matrix（属性矩阵）：打开属性矩阵面板，以电子表格的形式编辑可用的属性。

•Visibility Filter（可见性过滤器）：这允许你显示或隐藏已修改的属性或高级属性，以及合并或展开所有类目。

•Collapsible Categories（可合并的类目）：这些区域用于把相关的属性分组组合到一起，并且可以使用类目名称左侧的小白色三角形来展开及合并这些区域。

所有这些区域都是情境关联的。当这些区域中的其中一个区域不适合当前选中的项时，那个区域将会消失。比如，对于很多蓝图节点来说，看到基本上是空白的详细信息面板是正常的。

5.10

# 搜索结果面板

## 5.10.1　概述

Find Results（搜索结果）面板是蓝图编辑器中的一个强大的搜索工具，允许你快速地基于以下条件来追踪各种对象：

•Node name（节点名称）。

•Pin name（引脚名称）。

•Node Comment（节点注释）。

•Property name（属性名称）。

•Property value（属性值）。

当搜索结果面板跟踪搜索匹配项时，它将显示一个结果列表，每项结果就像一个超链接，可以让图表视图跳转到对应的节点上。当需要跟踪可能深埋于复杂的蓝图脚本节点网络中的特定节点或一段信息时，这是一种非常好的方法。

和在网页浏览器中搜索类似，当在蓝图编辑器中工作时，按下Ctrl+F组合键可以调出搜素结果面板。默认情况下，该面板出现在图表面板的底部。如果Compiler Results（编译结果）面板正在显示中，那么搜索结果面板将停靠在它的旁边。

和UE4中的很多搜索文本框不同，当你输入文本时，搜索结果面板不会动态地过滤结果，因为这可能会出现大量的结果。一旦你按下回车键后，搜索结果列表将会出现。

## 5.10.2　界面

界面如图5-19所示。

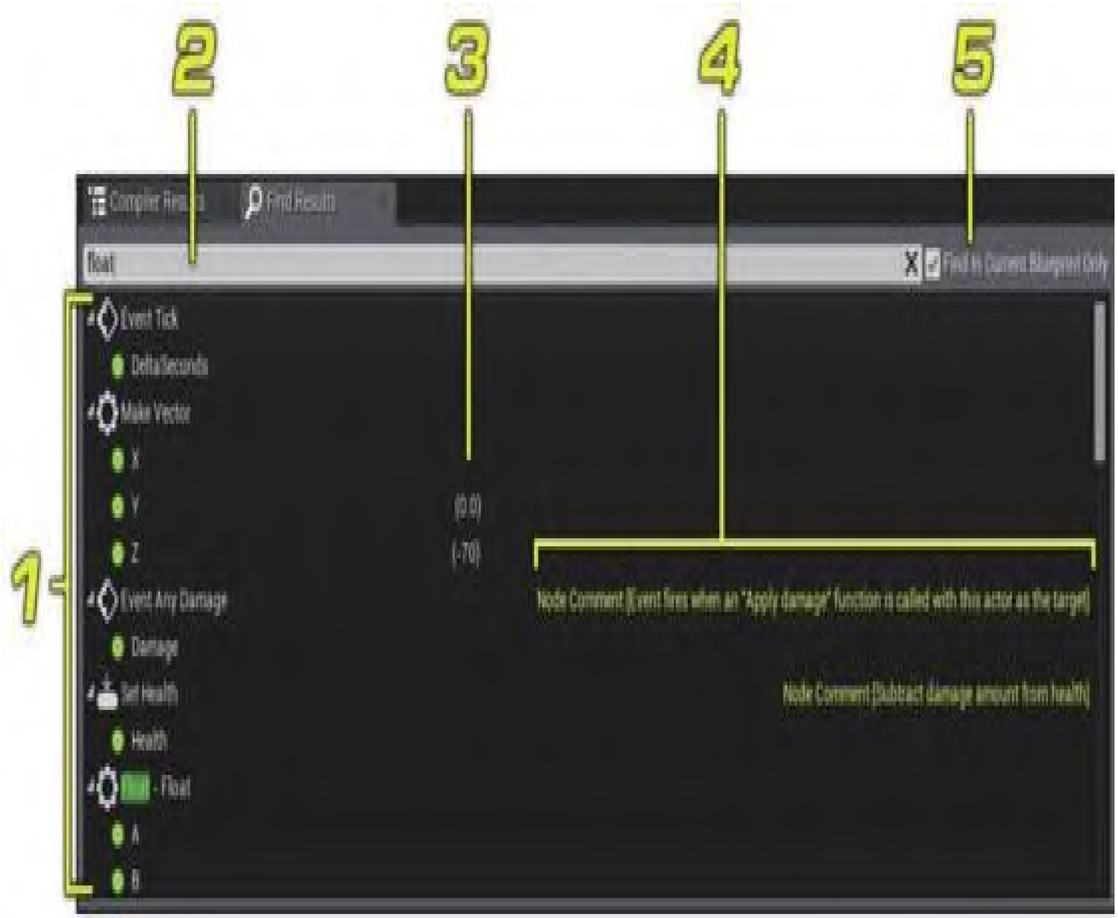



图5-19

•Results list（结果列表）：这里列出了和搜索条件匹配的所有节点、引脚、属性名称、注释及属性值。

•Search filter（搜索过滤器）：这是你输入你要查找的信息的地方。

•Property values（属性值）：明确设置的属性值出现在结果中间位置处的圆括号内。

•Comments（注释）：如果存在节点注释，那么它们以黄色文本的形式出现在面板的右侧。

•Find in Current Blueprint Only（仅在当前蓝图中搜索）：当启用该项时，搜索仅限于当前蓝图。当禁用该项时，搜索时会查找项目中的所有蓝

图。

5.11

# 菜单

## 5.11.1　概述

菜单默认显示在蓝图编辑器顶端。

## 5.11.2　文件

| 命　令 | 描　述 |
| --- | --- |
| 加载及保存 | |
| 保存 | 保存蓝图 |
| 保存所有 | 保存所有未保存的关卡和资源 |
| 选择要保存的文件 | 对内容和关卡打开带保存选项的对话框 |
| 提交到源代码控制 | 如果启用了源代码控制，对内容和关卡打开带迁入选项的对话框 |
| 蓝图 | |
| 编译 | 编译蓝图 |
| 刷新所有节点 | 刷新图表中的所有节点来应对外部变更 |
| 重定义蓝图父类 | 改变打开的蓝图的父类 |
| 比较文件差异 | 与上一版本比较差异，需要启用源文件控制 |
| 开发者 | 打开开发者菜单，你可以改变编译器设置，并且犹如在图表编辑器中一样，重新编译模块 |

### 5.11.3　编辑

| 命　令 | 描　述 |
| --- | --- |
| 历史记录 | |
| 取消 | 取消上一次执行的操作 |
| 重复 | 重复上一次取消的操作 |
| 搜索 | |
| 查找 | 在当前蓝图中查找函数，事件，变量和引脚的引用 |
| 在蓝图中查找 | 在所有蓝图中查找函数，事件，变量和引脚的引用 |
| 删除未使用的变量 | 删除任何从未使用的变量 |
| 设置 | |
| 编辑器偏好设置 | 打开编辑器设置 |
| 项目设置 | 打开当前项目设置 |

## 5.11.4　资源

| 命　令 | 描　述 |
| --- | --- |
| 在内容浏览器中查找 | 调出内容浏览器并导航到该资源 |

## 5.11.5　视图

| 命　令 | 描　述 |
|---|---|
| 引脚可见度 | |
| 显示所有引脚 | 显示所有节点的所有引脚 |
| 隐藏未使用的引脚 | 隐藏没有连接和默认值的所有引脚 |
| 隐藏未连接的引脚 | 隐藏所有未连接的引脚，此选项将隐藏节点上直接设置的输入的引脚 |

续表

| 命　令 | 描　述 |
|---|---|
| 缩放 | |
| 缩放至窗口 | 将当前视图缩放至整个图表 |
| 缩放至选中内容 | 将当前视图缩放至选中内容 |

### 5.11.6　调试

| 命　令 | 描　述 |
|---|---|
| 断点 | |
| 禁用所有数据断点 | 禁用当前蓝图或关卡蓝图的所有图表的所有数据断点 |
| 启用所有数据断点 | 启用当前蓝图或关卡蓝图的所有图表的所有数据断点 |
| 删除所有数据断点 | 删除当前蓝图或关卡蓝图的所有图表的所有数据断点 |
| 查看项 | |
| 删除所有查看项 | 删除当前蓝图或关卡蓝图的所有图表的所有查看项值 |

### 5.11.7　窗口

蓝图编辑器的窗口菜单有特定的子部分来显示蓝图编辑器的选项卡。当蓝图编辑器处于默认和组件模式时，菜单中并不会出现所有选项卡。

| 命　令 | 描　述 |
| --- | --- |
| 工具栏 | 如当前工具栏不可见，则显示工具栏 |
| 详细信息 | 如果详细信息面板当前不可见，则显示它 |
| 调试 | 如当前不可见，则显示调试面板 |
| 蓝图默认值 | 如当前不可见，则显示蓝图默认值面板 |
| Palette（面板） | 如果 Palette（面板）面板当前不可见，则显示它 |
| 我的蓝图 | 如果我的蓝图面板当前不存在则显示它 |
| 编译器结果 | 如当前不可见，则显示编译器结果面板 |
| 查找结果 | 如当前不可见，则显示查找结果面板 |
| 视口 | 如当前不可见，则显示预览视口面板 |

5.12

# 我的蓝图

我的蓝图显示了蓝图中的图表、脚本、函数、宏等的树形列表。这实质上是该蓝图的大纲视图，使你可以轻松地查看蓝图的现有元素及创建新的元素。

不同类型的蓝图在我的蓝图选卡的树形列表中显示不同类型的项目，如图5-20所示。

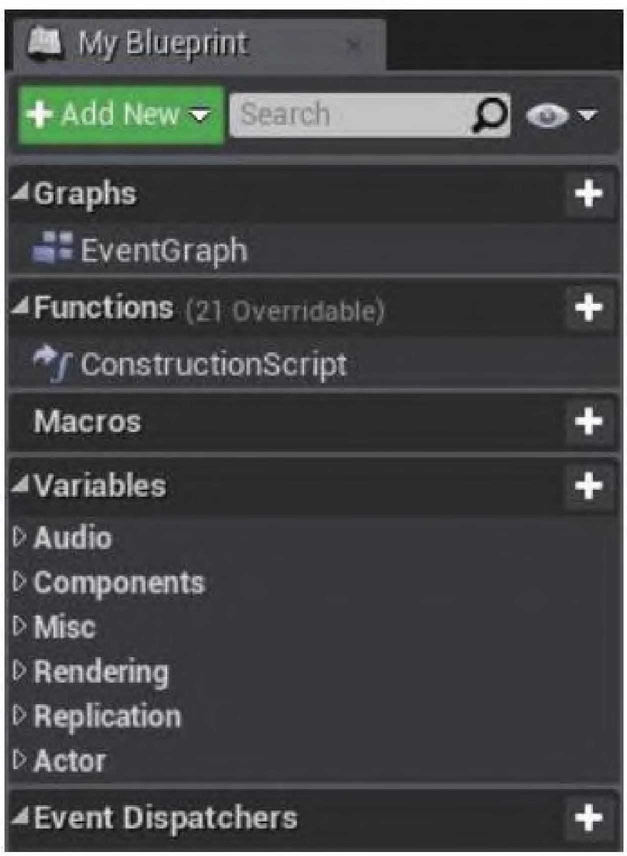


图5-20

比如，一个正常的蓝图将总是具有一个ConstructionScript和EventGraph。另外，还会显示在蓝图中创建的所有函数。一个关卡蓝图将仅具有一个EventGraph和在其内部创建的所有函数。一个Interface（接口）将仅显示在其内部创建的函数。一个Macro Blueprint（宏蓝图）将显示在其内部创建的所有宏函数。

## 5.12.1 创建按钮

我的蓝图选卡有一些用于创建新变量、函数、宏、事件图表及事件调度器的快捷按钮（每个标题末尾处的 + 图标）。

| 按　钮 | 名　称 | 描　述 |
|---|---|---|
| Variables + | 新建变量 | 当单击该按钮时创建一个新的变量。变量的属性会立即出现在详细信息选卡中。在那里，你可以改变变量的名称、类型及其他属性 |
| Functions + | 新建函数 | 创建一个新函数，然后立即将光标聚焦到详细信息选卡的Name（名称）文本域，以便可以对其进行命名。同时将会打开一个新的图表视图，在这里你可以定义该函数的节点网络 |
| Macros + | 新建宏 | 创建一个新宏，然后立即将光标聚焦到详细信息选卡的Name（名称）文本域，以便可以对其进行命名。同时将会打开一个新的图表视图，在这里你可以定义该宏的节点网络 |
| Graphs + | 新事件图表 | 创建一个新函数，然后立即将光标聚焦到详细信息选卡的Name（名称）文本域，以便可以对其进行命名。新的事件图表将会出现，且可以在其内部定义节点网络 |
| Event Dispatchers + | 新事件调度器 | 创建一个新事件调度器，然后立即将光标聚焦到详细信息选卡的Name（名称）文本域，以便可以对其进行命名 |

这些按钮可以通过在我的蓝图选卡中右击来进行访问。右击菜单还包含一个用于创建新的枚举值资源的选项，如图5-21所示。

最后，你可以在我的蓝图面板的顶部处使用添加新项目按钮，如图5-22所示。

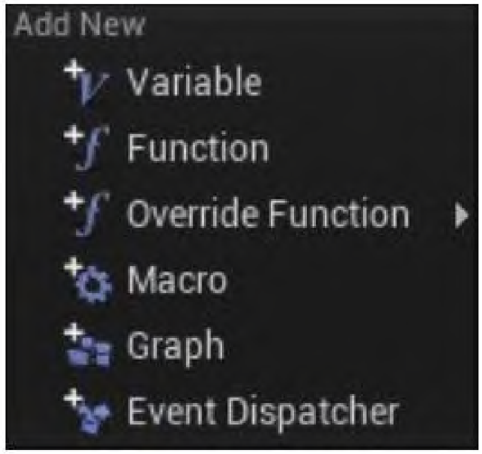



图5-21

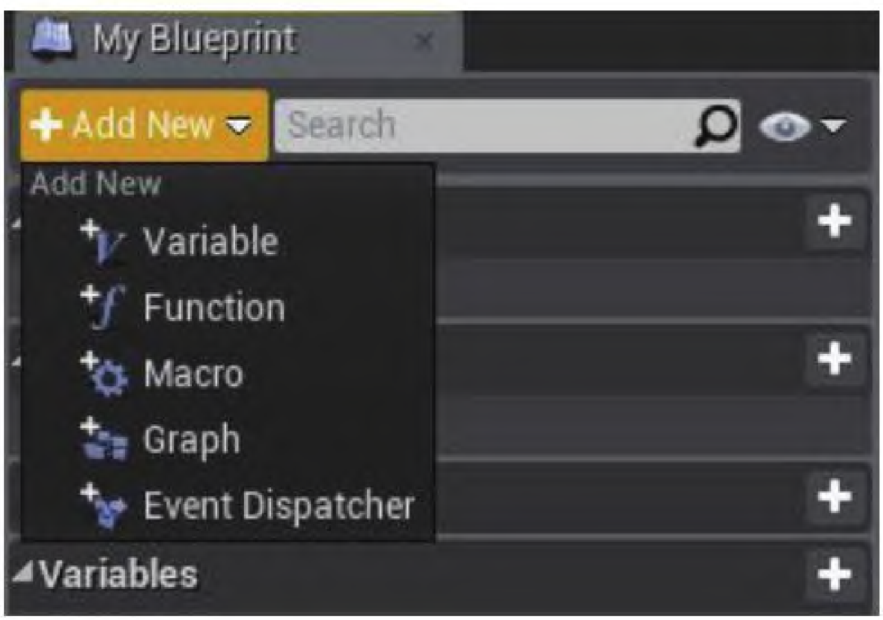



图5-22

## 5.12.2　我的蓝图部分

我的蓝图选卡分为6个部分：添加新项目、图表、函数、宏、变量以及事件调度器部分，如图5-23所示。

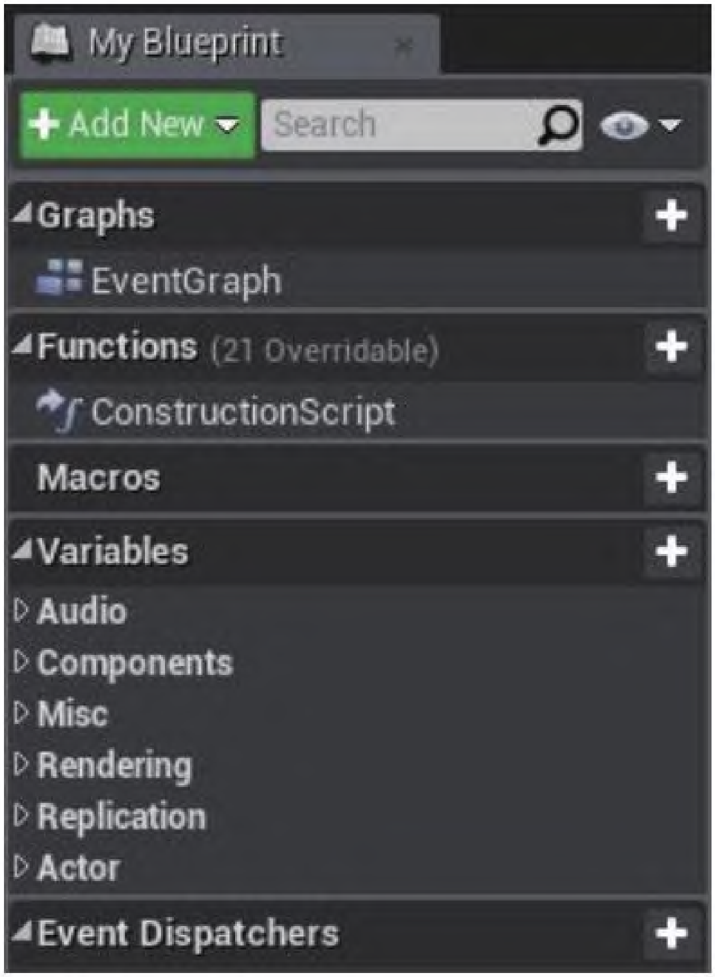



图5-23

面板下方的5个部分将会把你的蓝图元素整理到其相应的组中。顶部的部分让你可以快速创建新图表、变量等，并能搜索整个我的蓝图面板。

## 5.12.3　在我的蓝图中进行搜索

我的蓝图选卡包含一个文本框，用于搜索属于该蓝图的图表。这和你添加新节点所使用的关联动作菜单的工作方式类似，但仅限于在我的蓝图中搜索项目。你可以按名称、注释及其他数据进行搜索。因此，如果你知道你有一个节点叫SetActorHidden，那么你可以在那个文本框中输入该节点名称，浏览器将会显示你的图表中所有包含SetActorHidden的节点。

5.13

## Palette（面板）

Palette（面板）选卡显示了蓝图中可以使用的所有函数及变量的列表，而不是像图表选卡那样在关联菜单中显示过滤后的列表，如图5-24和图5-25所示。

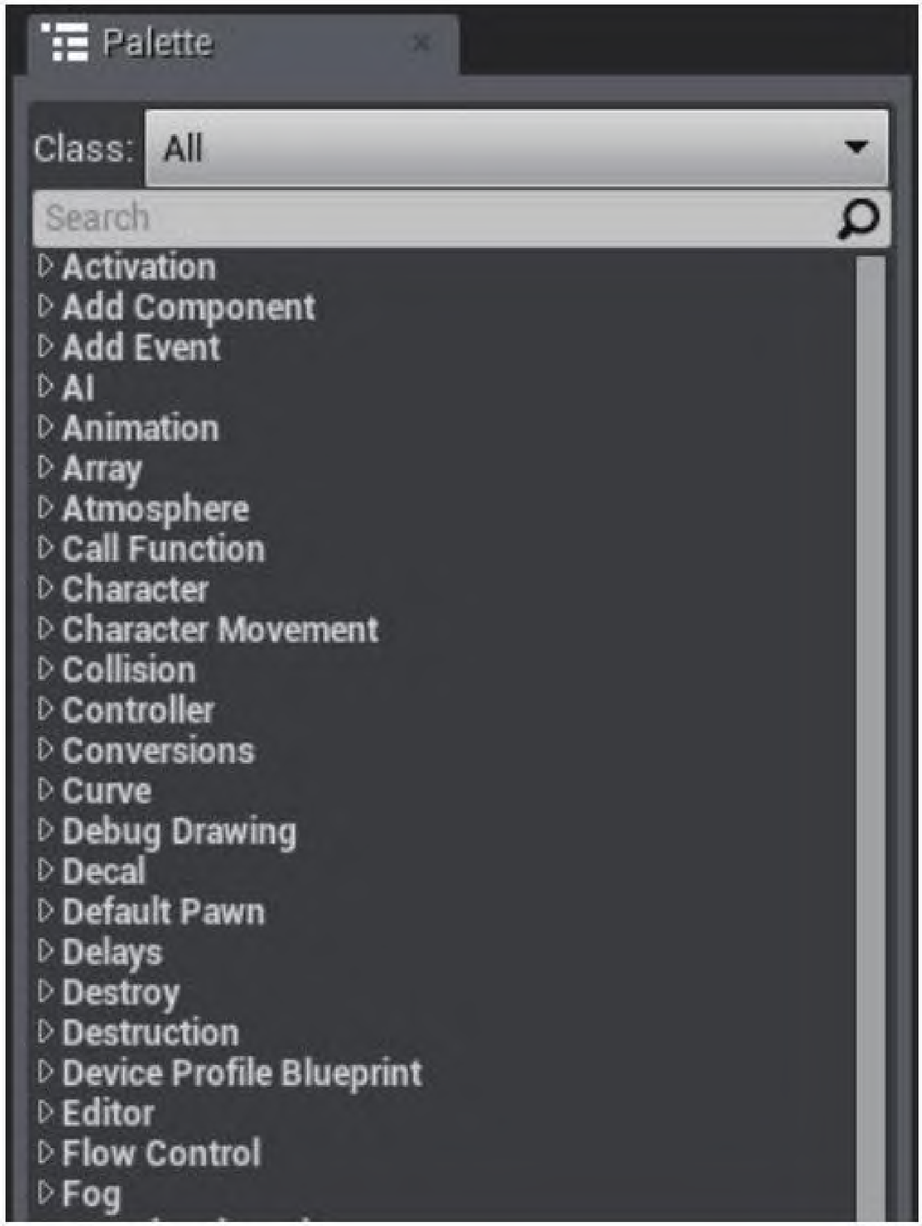

Palette
Class: All
Search
Activation
Add Component
Add Event
AI
Animation
Array
Atmosphere
Call Function
Character
Character Movement
Collision
Controller
Conversions
Curve
Debug Drawing
Decal
Default Pawn
Delays
Destroy
Destruction
Device Profile Blueprint
Editor
Flow Control
Fog

图5-24

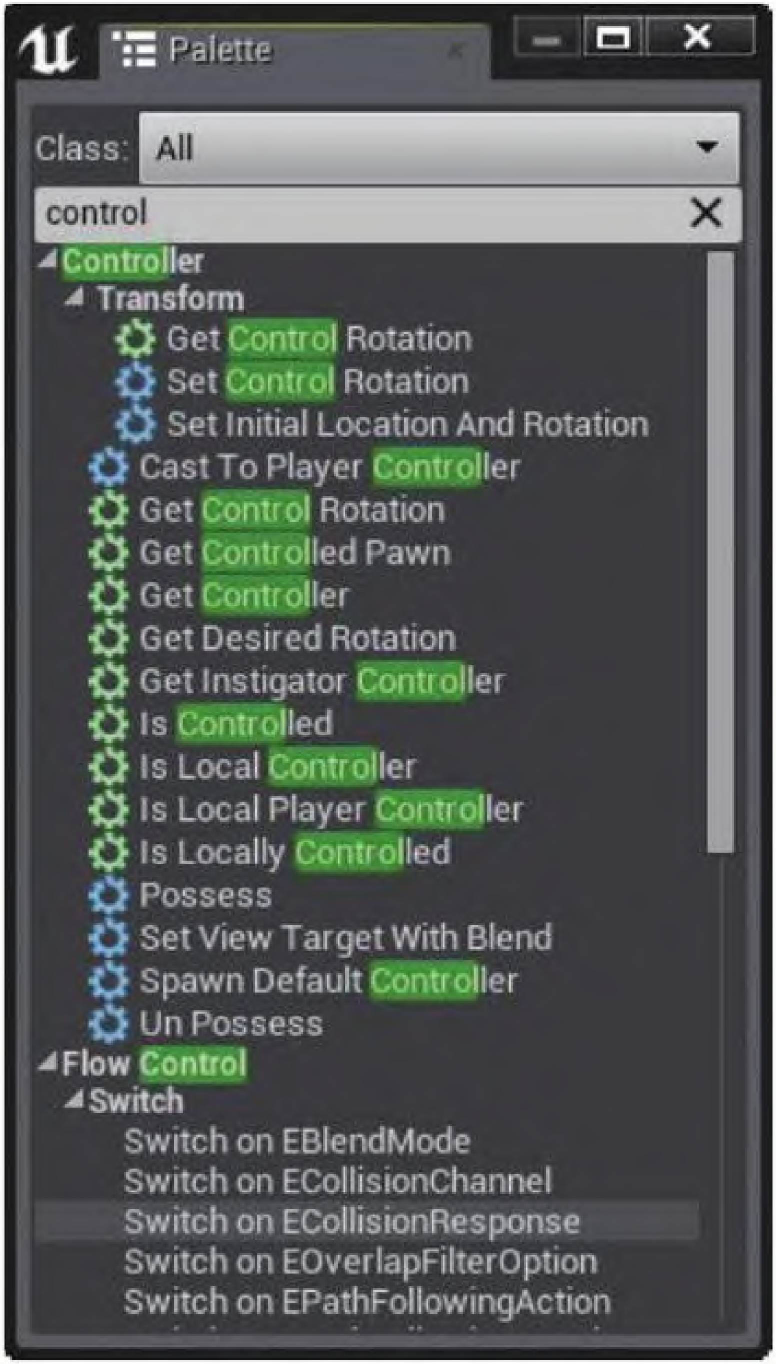



图5-25

和与情境关联的图表选卡菜单一样，Palette（面板）选卡底部提供了添加注释、自定义事件或时间轴的选项。

### 5.13.1 提炼面板结果

通过使用顶部的Search（搜索）框，可以过滤可用的函数和变量的列表。过滤列表使得可以更轻松地找到和特定主题相关的所有函数或变量。

比如，输入“control”可以找到和控制玩家或其他Actor相关的节点，如图5-26所示。

通过使用Palette（面板）选卡顶部的下拉菜单，可以仅显示和特定类相关的节点，如图5-27所示。

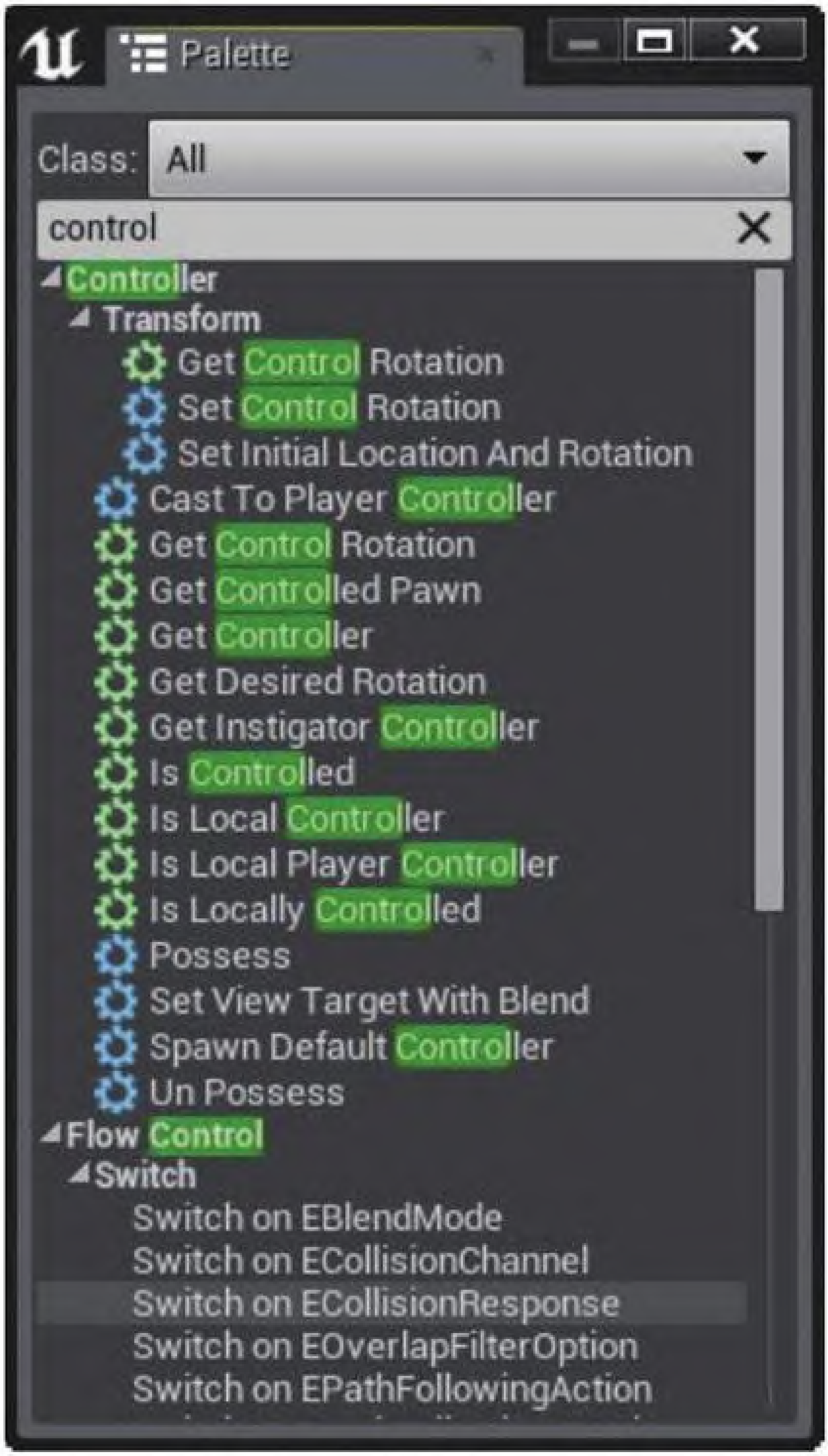

Palette
Class: All
control
Controller
Transform
Get Control Rotation
Set Control Rotation
Set Initial Location And Rotation
Cast To Player Controller
Get Control Rotation
Get Controlled Pawn
Get Controller
Get Desired Rotation
Get Instigator Controller
Is Controlled
Is Local Controller
Is Local Player Controller
Is Locally Controlled
Possess
Set View Target With Blend
Spawn Default Controller
Un Possess
Flow Control
Switch
Switch on EBlendMode
Switch on ECollisionChannel
Switch on ECollisionResponse
Switch on EOverlapFilterOption
Switch on EPathFollowingAction

图5-26

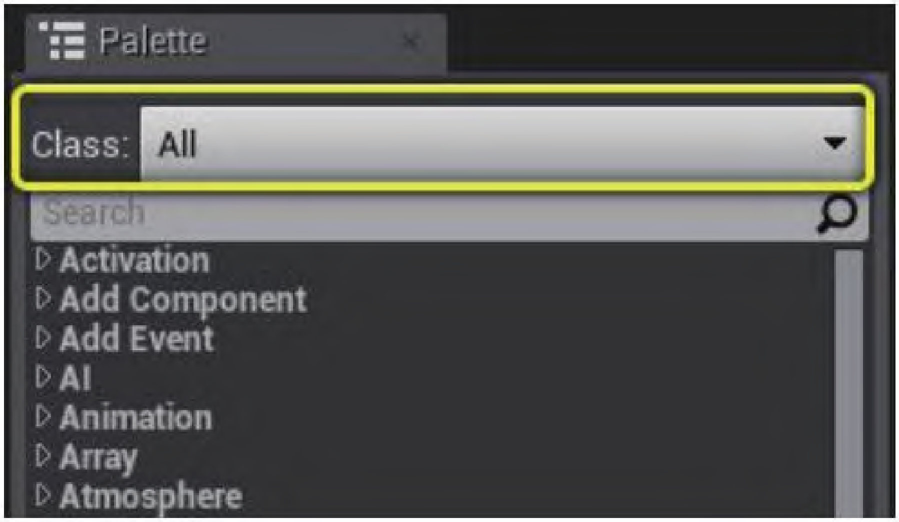



图5-27

## 5.13.2　放置节点

Palette（面板）选卡中的项目可以直接放置到图表中，通过把该项目从Palette中拖动并放置到图表中来实现。

| 拖动该项到图表中 | 放置该项来创建节点 |
| --- | --- |

5.14

# 蓝图编辑器视口

## 5.14.1　概述

在处于组件模式的蓝图编辑器视口中，你可以查看及操作你的蓝图的组件，如图5-28所示。

例。

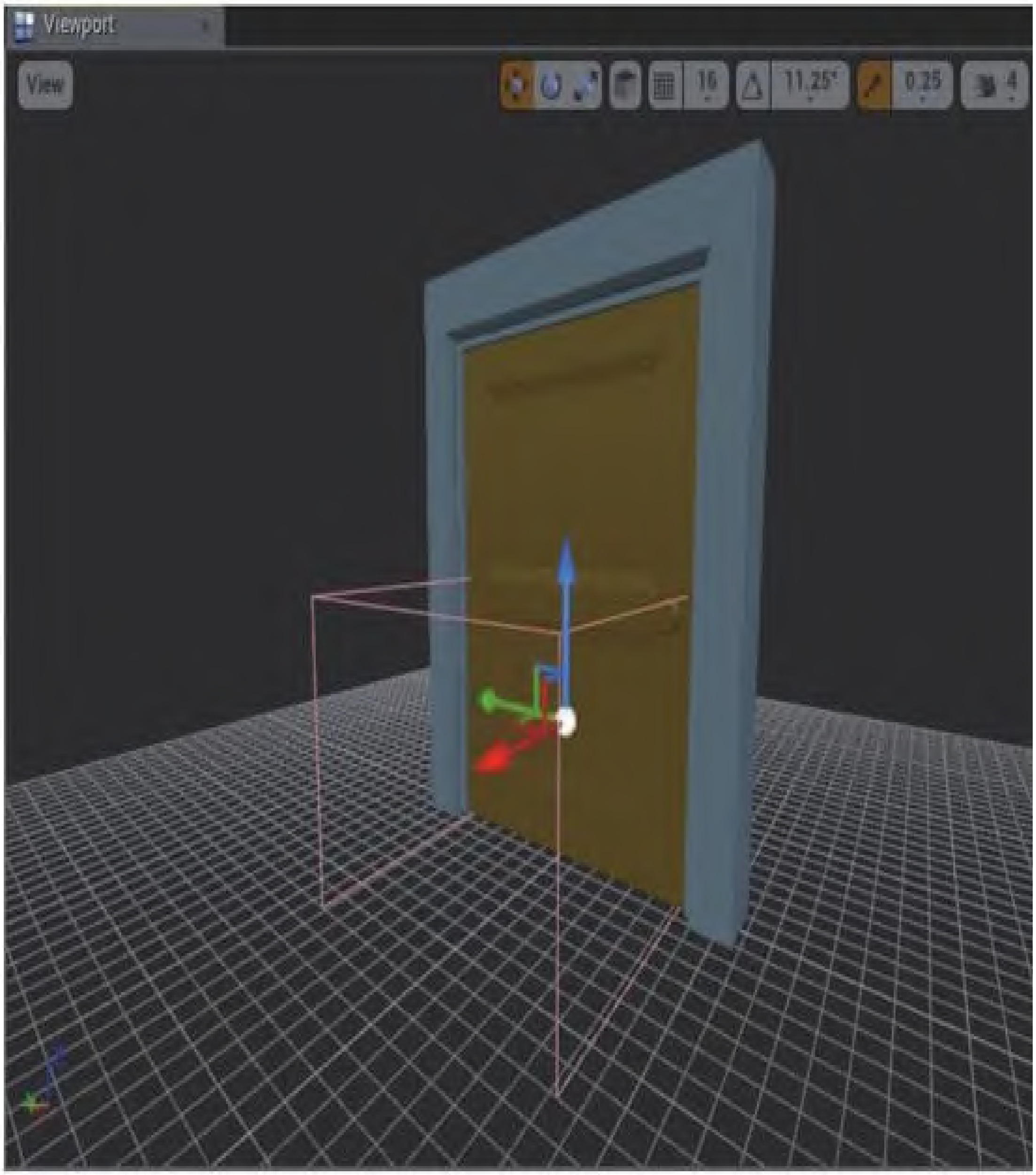



图5-28

你还可以通过在详细信息面板中通过手动地调整选中的组件的值来操作该组件。

在组件列表中选中一个组件，也会导致在视口中选中该组件，且详细信息面板中会显示该组件的属性信息。

你可以在视口中使用变换控件来调整选中组件的位置、旋转度及缩放比例。

在蓝图编辑器的视口中的导航方式同在关卡编辑器的视口中的导航方式一样。

## 5.14.2　视图菜单

位于视口的左上角的视图下拉菜单中有一些和关卡编辑器的视口菜单不同的选项，如图5-29所示。

图5-29

| 项　目 | 描　述 |
| --- | --- |
| Reset Camera（重置相机） | 重新让相机聚焦到组件上 |
| Realtime（实时渲染） | 切换视口的实时渲染功能 |
| Show Floor（显示地面） | 切换是否显示一个地面，以便辅助放置组件 |
| Show Grid（显示网格） | 切换视口网格 |

第6章

# 蓝图技术指南

蓝图是UE4中引入的一个强大的新功能。简单地讲，蓝图是创建新的UClasses的一种方法，通过这种方法，你不需要书写或编译任何代码。当你创建一个蓝图时，你可以选择继承“原生”C++类或者继承另一个蓝图类。然后，你可以添加、排列及自定义组件，使用可视化的脚本语言实现自定义的逻辑，对事件及交互做出反应，定义自定义变量，处理输入，及创建一种完全自定义的对象类型。

每个蓝图都具有一个“构建脚本”，和C++中的构造函数类似，当创建对象时会运行它。该脚本可以动态地基于多个因素构建Actor实例，比如，一个可以自动调整大小来填充建筑物间空隙的篱笆。从这个角度来说，蓝图也可以看成一个非常强大的预制系统。

6.1

# 暴露游戏元素给蓝图

## 6.1.1　概述

作为程序员，使用蓝图可以使你的代码非常灵活，这会为你带来很多好处。比如，游戏设计人员可能想在游戏中实现一种新类型的武器。你作为程序员，可以像以前一样编写武器的相关代码，但是你需要暴露一些重要的功能，比如设计速率和Fire（）函数。测试后，设计人员可以决定是否需要修改枪支的射击速率，以便在曲线上应用。这时不需要重新编码该枪支的射击速率并重新编译游戏，设计人员可以简单地到蓝图中直接地修改射击速率，这样既可以节省设计人员的时间，也可以节省程序员的时间。

## 6.1.2　创建蓝图可以继承的类

为了创建继承于一个类的蓝图，那么该类必须定义为Blueprintable（可蓝图化的），这涉及在处理类定义的UCLASS（）宏中添加这个关键字。该关键字使得蓝图系统知道这个类的存在，以便该类显示在New Blueprint（新建蓝图）对话框的类列表中，并且可以选择它作为正在创建的蓝图的父类。

声明蓝图可以继承的类的最简单的格式如下所示：

```
UCLASS(Blueprintable)
class AMyBlueprintableClass : AActor
{
    GENERATED_UCLASS_BODY()
}
```

| 关键字 | 描述 |
|---|---|
| Blueprintable | 暴露这个类为创建蓝图时可以接受的基类。默认为是NotBlueprintable，除非从其他类继承Blueprintable关键字，子类可以继承这个属性 |
| BlueprintType | 暴露这个类作为蓝图中变量可以使用的一种类型 |
| NotBlueprintable | 指定这个类是不能作为创建蓝图的基类。使得父类具有的Blueprintable关键字无效 |

## 6.1.3　可读和可写属性

为了将在C++类中定义的一个变量暴露给继承该类的蓝图，那么那个变量在处理变量定义的UPROPERTY（）宏中必须使用以下列出的其中一个关键字进行定义。这些关键字使得蓝图系统意识到该变量的存在，从而使它显示在MyBlueprint（我的蓝图）面板中，并且可以设置或访问它的值。

```
//Character's Health
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category="Character")
float health;
```

| 关键字 | 描述 |
| --- | --- |
| BlueprintReadOnly | 该属性由蓝图读取，但是不能修改 |
| BlueprintReadWrite | 该属性可以从蓝图中读取或写入 |
| 多路广播代理关键字 | |
| BlueprintAssignable | 暴露该属性来在蓝图中进行分配赋值 |
| BlueprintCallable | 暴露该属性来在蓝图图表中进行调用 |

## 6.1.4 可执行和可重载的函数

为了从蓝图中调用一个原生函数，则必须在处理函数定义的UFUNCTION（）宏中使用下表列出的其中一个关键字来定义函数。这些关键字使得蓝图系统意识到该函数的存在，以便它可以显示在关联菜单或者Palette（控制板）中，可以添加到图表中并执行（如果它们是事件），以便可以进行重载及执行。

声明BlueprintCallable（蓝图可调用）函数的最简单的形式如下所示：

```
//Fire a Weapon
UFUNCTION(BlueprintCallable, Category=" Weapon" )
void Fire();
```

| 关键字 | 描述 |
| --- | --- |
| 蓝图到原生代码的通信 | |
| BlueprintCallable | 这是一个可以从蓝图中调用的原生函数，它执行原生代码，修改正在调用它的对象上的一些内容或者一些其他的全局状态。这意味着它应该是“提前规划的”，或者已经清楚地知道相对于其他节点的执行顺序。我们通过白色执行连线完成这个处理。所有蓝图可调用的函数都按照它们沿着白色执行线出现的顺序进行调用 |
| BlueprintPure | 这是一个可以从蓝图中调用的原生函数，它执行原生代码，不会修改调用它的对象的任何内容，也不会修改任何其他的全局状态。这意味着调用该代码不会修改任何内容，它仅是取入输入，并向你提供一个输出。像数学节点（+、-、*等）、变量获取节点，或者任何不永久性地修改任何内容的东西都属于这类内容。这些内容不需要规划如何执行，它们没有线连接到白色执行线。编译器会根据哪个 BlueprintCallable 节点需要这些节点处理的数据来自动地辨别出它们 |

续表

| 关键字 | 描述 |
| --- | --- |
| 原生代码到蓝图的通信 | |
| BlueprintImplementableEvent | 这是我们允许原生函数调用蓝图的主要方式。它们就像你在蓝图本身内实现的虚函数。如果函数没有实现，那么该函数调用将会被忽略掉。需要注意的很重要的一点是，如果一个 BlueprintImplementableEvent 没有返回值或输出参数，那么它将呈现为一个事件，你可以通过右击并从蓝图的事件图表中选择它。如果它具有返回值或任何输出参数，那么它将列在 My Blueprints（我的蓝图）中，并且可以通过右击它并选择“实现”函数来重载它。注意，BlueprintImplementableEvents 没有该函数的原生代码实现 |
| BlueprintNativeEvent | 这些关键字和上面的一样，但是它具有该函数的原生默认实现，如果蓝图没有重载该函数则调用默认实现。如果蓝图没有实现函数，你需要某种默认行为；但是在需要重载函数的地方，你又需要重载该函数，对于这种情况这个关键字是非常有用的。这些关键字的性能消耗较大，所以我们仅把它们放在需要该功能的地方。当你重载该 BlueprintNativeEvent 时，如果你想调用原生实现，那么通过右击该事件或函数入口节点并选择“Add call to parent”，你仍然可以调用原生实现 |

6.2

## 蓝图编程指南

可从两个主要方面考虑使用C++或蓝图：

•速度。

•表达式复杂度。

这两个因素之外，便主要取决于游戏的复杂程度和团队的构成。如果团队中美术师的数量多于程序员，则蓝图的优越性大于C++代码。相反如果程序员数量较多，则最好使用C++。还需要预计开发过程中人员的缺席情况。在Epic，普遍的工作流程是：内容设计师创建十分复杂的蓝图后，程序员编写新蓝图节点的代码，明确如何将大量工作压缩到C++中，然后将功能块移入一个新的C++函数。较好的方法是大量使用蓝图，复杂度达到一定程度，启用功能更简洁的表达式（否则它对非程序人员而言过于复杂）时，可将内容转入C++，或由执行速度指定到C++的移动。

### 6.2.1　速度

实际上，蓝图执行的速度比C++执行慢。这并不意味着会影响性能。但需要大量计算或进行高频率操作时，最好使用C++。也可以根据团队状况和项目性能对二者进行完美结合。如蓝图功能较多，可将部分功能移入C++以使速度加快，蓝图中剩余的部分功能可保留灵活性。如分析表明蓝图中的一项操作耗时极长，可考虑只将该部分移入C++，其他部分仍然保留在蓝图中。

举例而言，控制上千个Actor的群组系统通过蓝图可视化脚本执行时间极长。在这类情况下，在C++中处理决定、路径和其他群组功能性能更佳，之后可将部分调整参数和控制函数公开到蓝图。

### 6.2.2　复杂度

就表达式复杂度而言，部分操作在C++中更容易执行。蓝图可良好地执行大量操作，但部分内容不易在节点中表达。操作大型数据集、执行字符串操作，在大量数据集上执行复杂算术等内容均十分复杂，不易在视觉系统中进行操作。这些内容更适合于C++，因为可以轻松进行查看，了解具体细节。群组系统更适合C++代码的另一个原因在于表达式复杂度。

### 6.2.3　范例

部分功能块适合在C++中处理，部分适合在蓝图中处理。所以在此列举一些游戏开发中C++程序员和蓝图设计者搭档工作的实例。

•程序员可在C++中创建定义自定义事件的角色类，之后蓝图可用于该角色类的延展，实际指定模型并进行默认设置。查看ShooterGame范例项目中的玩家角色和敌方机器人，了解类似实现。

•可在C++中实现能力系统的基础类，之后设计师再创建实际执行操作的蓝图。在StrategyGame范例项目中有一个在C++中定义的基础炮台，但火焰喷射器、火炮台和弓箭台的行为均在蓝图中定义。

•“收集”或“重生”函数为可在蓝图中实现的事件，这样的可拾取道具可被覆写，以便设计师生成不同粒子发射器和声效。ShooterGame和StrategyGame中均有此方式创建的可拾取道具。

### 6.2.4　创建蓝图API：提示和技巧

程序员创建对蓝图公开的API时需要考虑以下几点：

•可选参数便于在蓝图中处理：

```
/**
 * 将字符串显示到日志中，也可选择显示到屏幕上。
 * 如Print To Log为true，它将显示在Output Log窗口中。否则它将被记录为"Verbose"，通常不会显示。
 *
 * @param   InString        登出字符串
 * @param   bPrintToScreen  是否将输出显示到屏幕上
 * @param   bPrintToLog     是否将输出保存到日志中
 * @param   bPrintToConsole 是否将输出显示到控制台
 * @param   TextColor       是否将输出显示到控制台
 */
UFUNCTION(BlueprintCallable, meta=(WorldContext="WorldContextObject", CallableWithoutWorldContext, Keywords = "log print", AdvancedDisplay = "2"), Category="Utilities|String")
static void PrintString(UObject* WorldContextObject, const FString& InString = FString(TEXT("Hello")), bool bPrintToScreen = true, bool bPrintToLog = true, FLinearColor TextColor = FLinearColor(0.0,0.66,1.0));
```

•在带大量返回参数的函数和返回结构体的函数之间优先前者。以下片段显示如何在节点上创建多个输出引脚：

```
UFUNCTION（BlueprintCallable, Category=“Example Nodes”）
static void MultipleOutputs（int32& OutputInteger, FVector& OutputVector）;
```

•可在现有函数上添加新参数，但如果要进行移除或变更，则需要否决原始函数并添加一个新函数。必须使用否决元数据，使新函数的信息显示在蓝图中。

```
UFUNCTION(BlueprintCallable, Category=" Collision" ,
meta=(DeprecatedFunction, DeprecationMessage = "Use new
CapsuleOverlapActors" , WorldContext=" WorldContextObject" , AutoCreateRe
fTerm=" ActorsToIgnore" ))
static ENGINE_API bool CapsuleOverlapActors_DEPRECATED(UObject*
WorldContextObject, const FVector CapsulePos, float Radius, float
HalfHeight, EOverlapFilterOption Filter, UClass* ActorClassFilter, const
TArray<AActor*>& ActorsToIgnore, TArray<class AActor*>& OutActors);
```

•如果函数需要接受枚举，考虑将“expand enum as execs”用作元数据，可使节点更易于使用。

```
UFUNCTION(BlueprintCallable, Category = "DataTable" , meta =
(ExpandEnumAsExecs=" OutResult" , DataTablePin=" CurveTable" ))
static void EvaluateCurveTableRow(UCurveTable* CurveTable, FName RowName, float
InXY, TEnumAsByte<EEvaluateCurveTableResult::Type>& OutResult, float& OutXY);
```

•许多完成耗时较长的操作（如move here）均为隐藏函数。

```
/**
 * 执行带延迟的隐藏操作。
 *
 * @param WorldContext  世界背景。
 * @param Duration      延迟长度。
 * @param LatentInfo    隐藏操作。
 */
UFUNCTION(BlueprintCallable, Category="Utilities|FlowControl", meta=(Latent, WorldContext="WorldContextObject", LatentInfo="LatentInfo", Duration="0.2"))
static void Delay(UObject* WorldContextObject, float Duration, struct FLatentActionInfo LatentInfo );
```

•如有可能，考虑将函数放入共享库。便于在多个类之间使用，避开“target”引脚。

```
class DOCUMENTATIONCODE_API UTestBlueprintFunctionLibrary : public UBlueprintFunctionLibrary
```

•尽可能将节点标记为纯，可避免在节点上使用连线的执行引脚。

```
/* 在 0 和 最大 - 1 之间返回一致分配的随机数 */
UFUNCTION(BlueprintPure, Category="Math|Random")
static int32 RandomInteger(int32 Max);
```

•将一个函数标记为const也可使蓝图节点不带执行引脚。

```
/**
 * 获得 actor 到世界的转换。
 * @return 从 actor 空间转换到世界空间的转换。
 */
UFUNCTION(BlueprintCallable, meta=(DisplayName = "GetActorTransform"),
Category="Utilities|Transformation")
FTransform GetTransform() const;
```

6.3

# 原生化蓝图

这是一个实验性beta功能。所有UE4游戏实例和部分中等大小的demo均已成功原生化，但我们仍在努力，力求在大型项目中使用此功能。

全新的蓝图原生化工具可从项目蓝图生成本地C++代码。完成项目后在编辑器外进行测试或进行发布时，将使用生成的代码对项目进行编译。所有蓝图类将被新生成的本地类替代。蓝图原生化的主要目的是降低项目运行时版本中的虚拟机（VM）开销。因此，生成的代码并未设计为“友好”或可重复使用。

此节将助你上手项目蓝图的原生化。对纯蓝图项目而言，这应该是透明度最高的一个流程。如项目中也使用了C++，或需要对原生化代码进行调试，请查阅高级蓝图原生化中的更多内容。

## 6.3.1　启用蓝图原生化

如需启用蓝图原生化，可使用Edit菜单打开Project Settings，在Packaging下启用Nativize Blueprint Assets复选框，如图6-1所示。

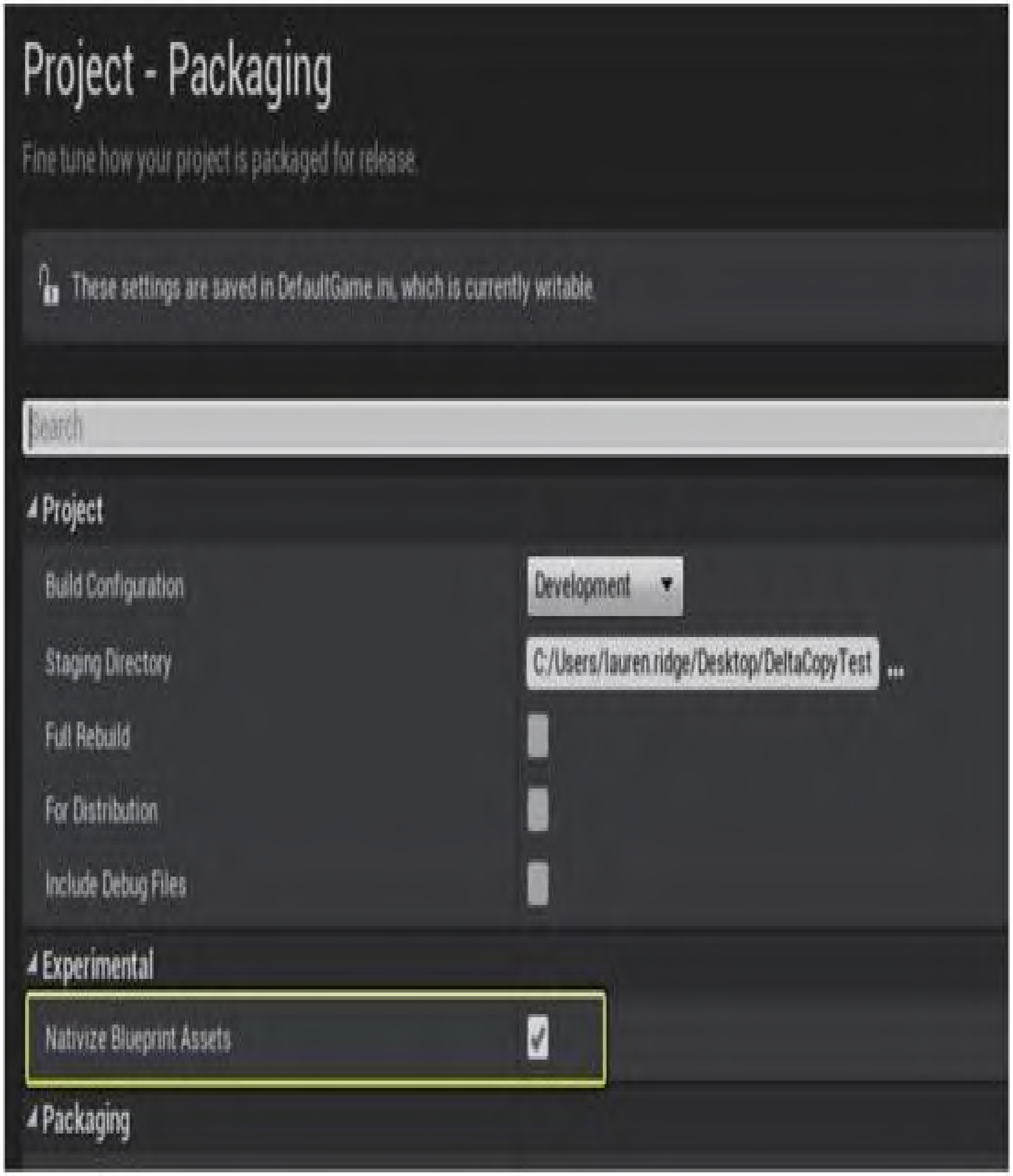



图6-1

之后打包项目时，以下蓝图资源将被原生化：

•蓝图类。

•蓝图函数库。

•蓝图接口。

•动画蓝图。

•控件蓝图。

•用户定义结构。

•用户定义枚举。

当前关卡蓝图将不会被转换。原生化代码保存的路径为（项目目录）\Intermediate（制作平台）\NativizedAssets\Source\NativizedAssets\。

### 6.3.2　从转换中排除蓝图

也可从转换中排除特定蓝图。在项目的DefaultEditor.ini文件中创建一个（BlueprintNativizationSettings）标题。

•如需排除特定类型的资源，可设置ExcludedBlueprintTypes，例如，+ExcludedBlueprint Types=/Script/UMGEditor.WidgetBlueprint不会转换控件蓝图。

•如需排除单个资源，可设置ExcludedAssets，例如，+ExcludedAssets=/Engine/Tutorial/SubEditors/TutorialAssets/Character/TutorialCharacter不会转换教程角色。

资源只有在被制成后才可进行原生化。如果未指定制作器制作资源/地图，它将对项目Content目录下的所有内容进行制作。如发现部分未使用的蓝图正被原生化，也可使用地图列表进行打包，排除资源，如图6-2所示。

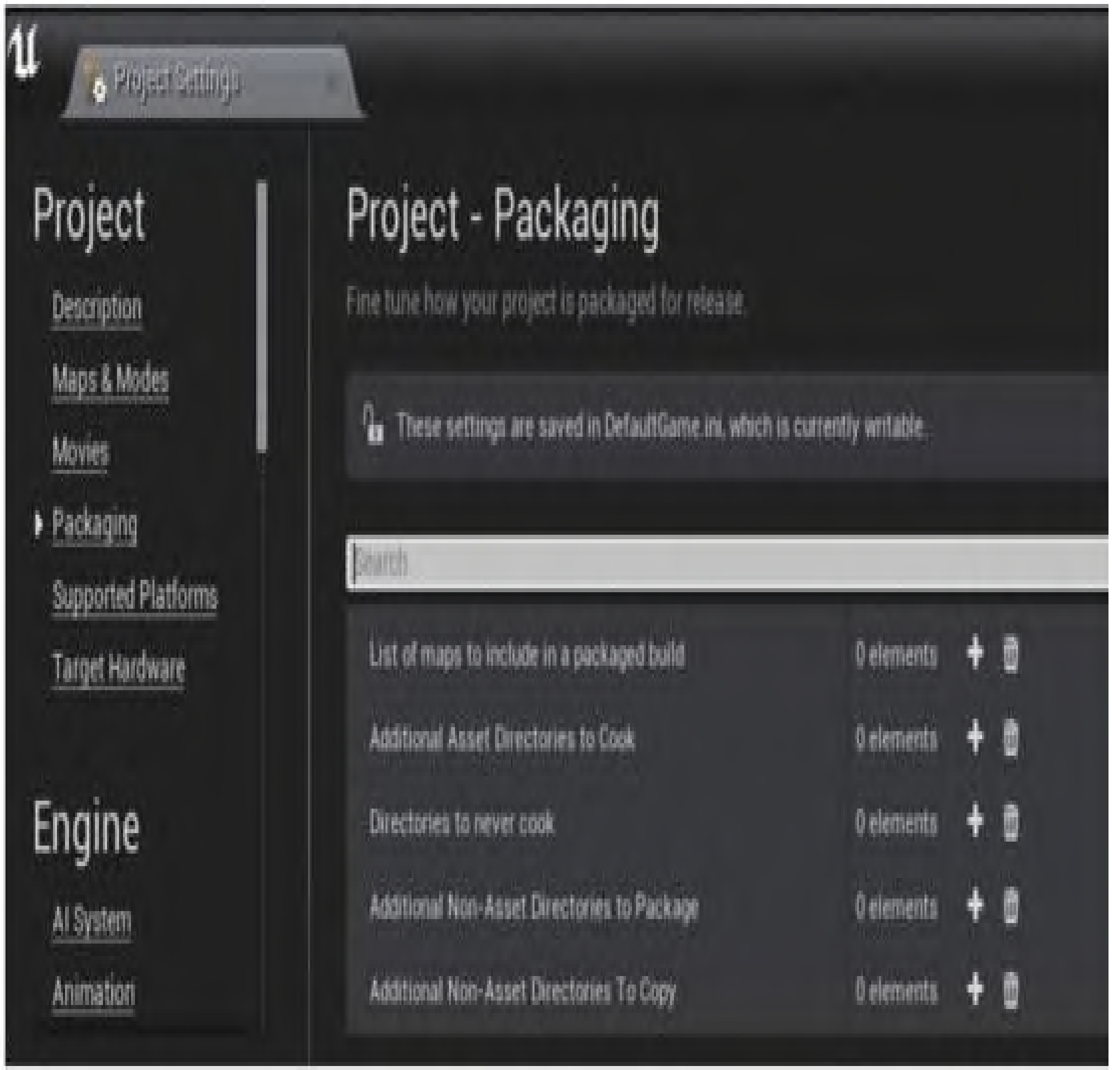

Project Settings
Project
Description
Maps & Modes
Movies
Packaging
Supported Platforms
Target Hardware
Engine
AI System
Animation
Project - Packaging
Fine tune how your project is packaged for release.
These settings are saved in DefaultGame.ini, which is currently writable.
Search
List of maps to include in a packaged build
0 elements
Additional Asset Directories to Cook
0 elements
Directories to never cook
0 elements
Additional Non-Asset Directories to Package
0 elements
Additional Non-Asset Directories To Copy
0 elements

图6-2

6.4

# 高级蓝图原生化

## 6.4.1　使用虚幻自动化工具进行构建

如要在使用虚幻自动化工具进行构建时原生化蓝图，须在BuildCookRun命令行中添加-nativizeAssets开关。

## 6.4.2　C++协作

原生化蓝图保存在单独的新模块中，意味着游戏性模块的头文件正在模块之外使用。须进行检查，确定所有头文件已将使用的所有类和结构体作为内含头文件或前置声明进行声明。

一些规则/限制有助于使手写代码和原生化代码之间的协作更加高效可靠。

不支持以下实现：

•带BlueprintVisible或Editable标记的私有/受保护变量。

•标记为BlueprintPure或BlueprintCallable的私有/受保护函数。

•noexport结构/类。

•带BlueprintVisible或Editable标记的位域变量。

•私有位域变量。

•UBlueprintGeneratedClass、UUserDefinedStruct、UUserDefinedEnum变量，或基于它们的任意类型。

## 6.4.3 技术细节

原生化蓝图兼容Visual Studio 2013和Visual Studio 2015，但强烈推荐使用2015版，因为它的大项目处理能力更强。如出现奇怪的编译器/连接器错误（如LNK1000、LNK1248），可尝试将部分类型或特定蓝图排除，削减原生化蓝图的量。

BlueprintCompilerCppBackend模块（此路径中：Engine\Source\Developer\Blueprint CompilerCppBackend\）将把蓝图语法树转入C++。BlueprintNativeCodeGen模块（此路径中：Engine\Source\Developer\BlueprintNativeCodeGen\）在烘焙和打包时可大显身手。

第7章

# 在线会话节点

7.1

# 在线会话节点

以下节点通过蓝图在多人游戏中创建主机、寻找、加入或退出。

这些节点为隐藏，意味着它们在后台执行操作（通常因为这些操作必须通过网络进行交流），并在之后的某点结束。节点右上角的钟表图标即代表其为隐藏节点。

7.2

## 创建会话

首先从Create Session节点开始。游戏成功创建一个会话后，游戏的其他实例即可发现（使用Find Session节点）并加入（使用Join Session节点）。在此节点上可指定会话中允许的玩家数量，以及是否可在LAN模式中创建会话。对于支持的在线子系统模块而言，将LAN模式设为false将尝试创建在线游戏，如图7-1所示。

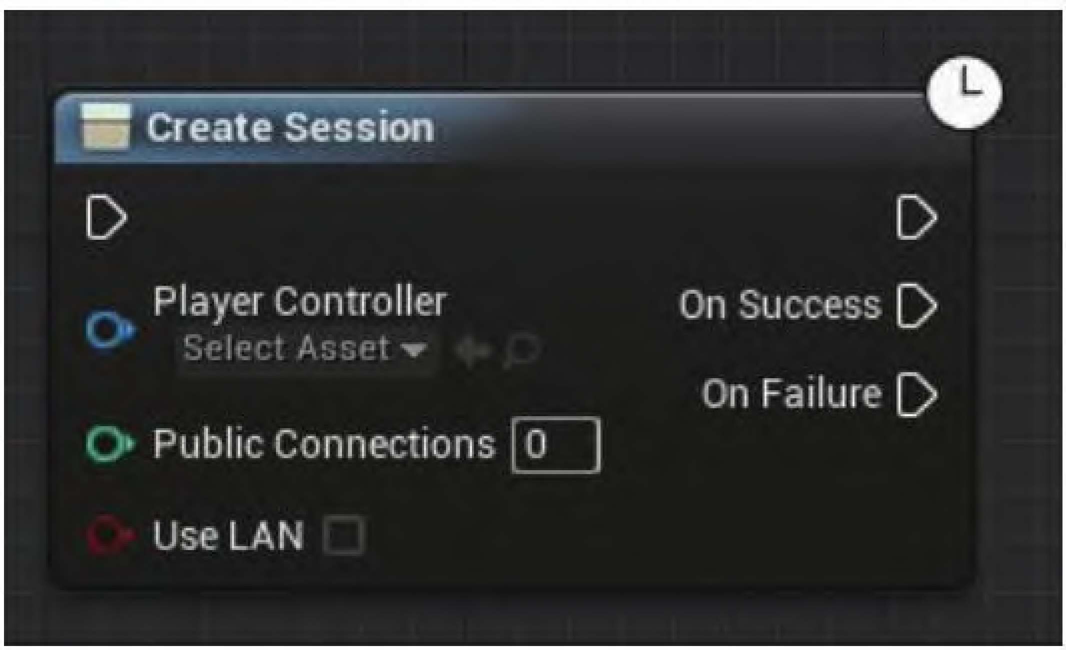



图7-1

7.3

## 寻找会话

使用Find Sessions节点获取当前公布并可加入的游戏会话的列表。BlueprintSessionResult从成功调用返回后可获取服务器的名称、ping和玩家数量，如图7-2所示。

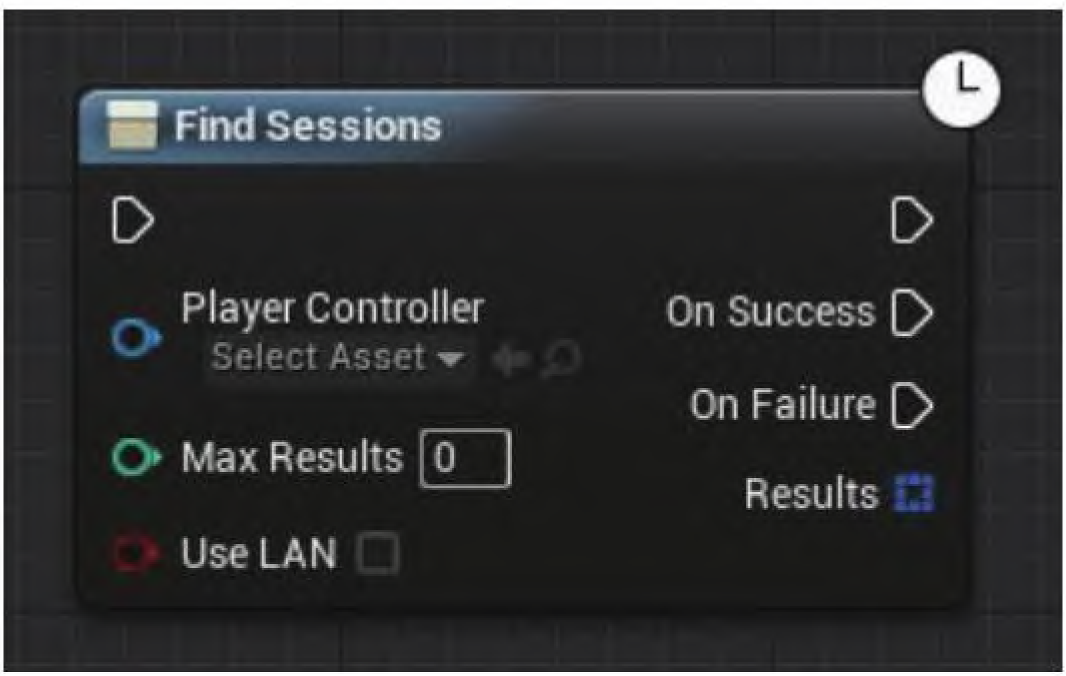



图7-2

7.4

## 加入会话

确定加入的BlueprintSessionResult后，将其传入JoinSession调用。如游戏成功连接到服务器，它将自动前往服务器的映射并加入游戏，如图7-3所示。

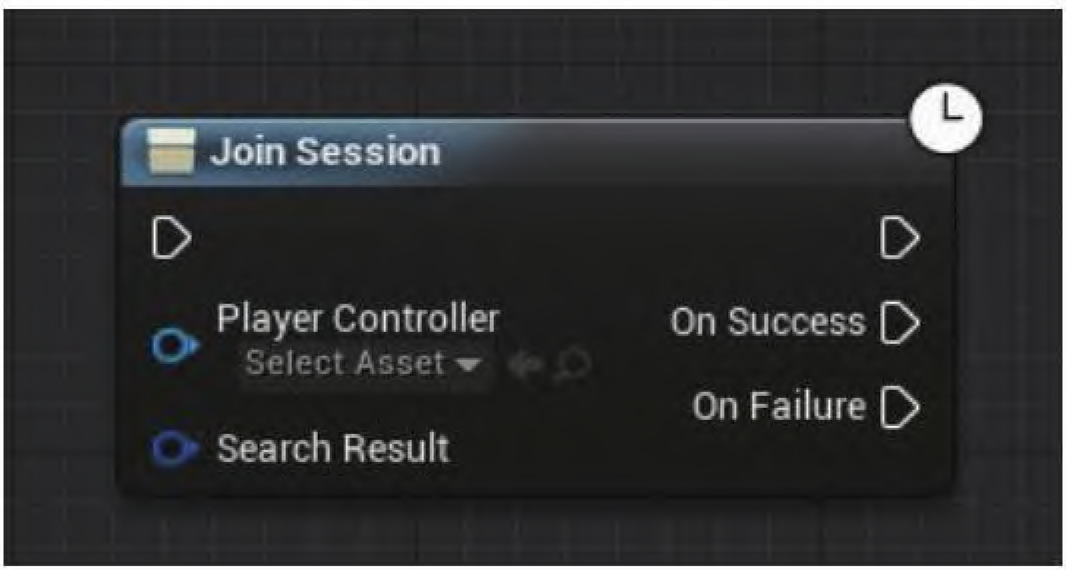



图7-3

7.5

## 销毁会话

如不希望其他玩家发现并加入游戏，可调用Destroy Session节点。此操作将导致你离开会话；如你为主机，将会话关闭，使其无法被发现，如图7-4所示。

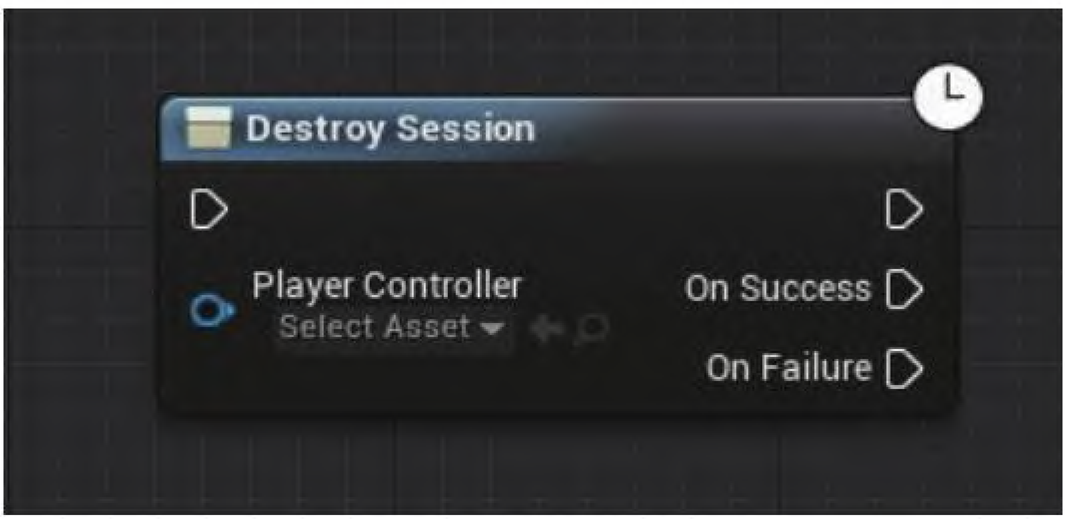



图7-4

此节点还用于离开会话。客机离开游戏时也需要调用此节点。

7.6

## 错误处理节点

处理网络操作时始终存在出现错误的可能，游戏需要对错误进行恰当的处理。可能主机已于先前推出你尝试加入的会话，也可能你的网络服务已临时关闭，还可能出现不胜枚举的各种其他错误。游戏可通过蓝图灵活处理这些与网络相关的错误。

以上会话操作出现的错误将通过OnFailure执行引脚进行报告，以便对游戏进行设置，处理问题。对于一些特定类型的错误，或游戏进程中发生的错误，可从GameInstance蓝图中调用两个错误事件Network Error和Travel Error。

7.7

## Event Network Error

Network Error事件允许游戏对事件进行回应，接收描述错误类型的枚举值，并进行适当的回应，如游戏可能对用户显示一个对话框，如图7-5所示。

图7-5

7.8

# Event Travel Error

Travel Error事件不及Network Error常见，可使游戏处理Network Error未涉及的情况（如尝试加载无效地图），如图7-6所示。

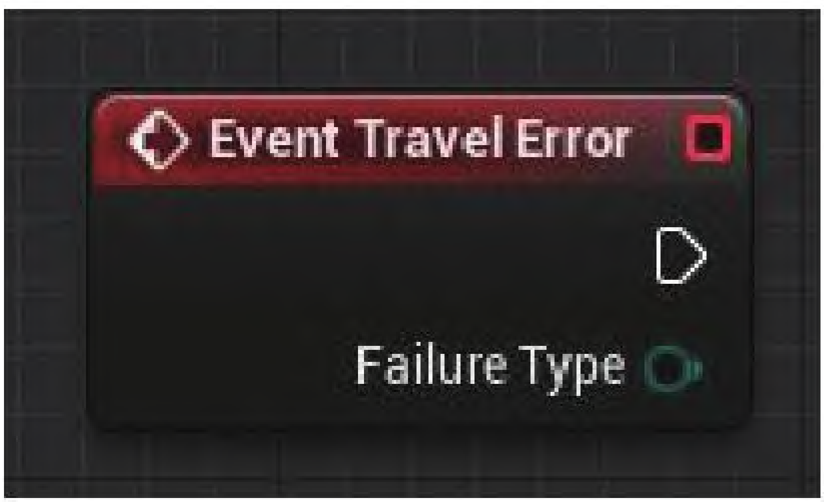



图7-6

第8章

# Bitmask蓝图变量

蓝图现支持将整数变量声明为位掩码。个体屏蔽位（标志）即可通过编辑器进行设置和复原，无须手动输入数值。此外，蓝图枚举可被声明为位掩码类型，用作显式位标记标识符。

8.1

## 创建位掩码变量

在蓝图编辑器中使用蓝图类时，任意整数变量均可作为位掩码创建并声明。

**Step 1** 勾选蓝图枚举编辑器中的Bitmask Flags属性创建同为位标记枚举类型的蓝图枚举。也可使用C++中通过BitmaskEnum=元数据设置的现成位标记枚举类型，如图8-1所示。

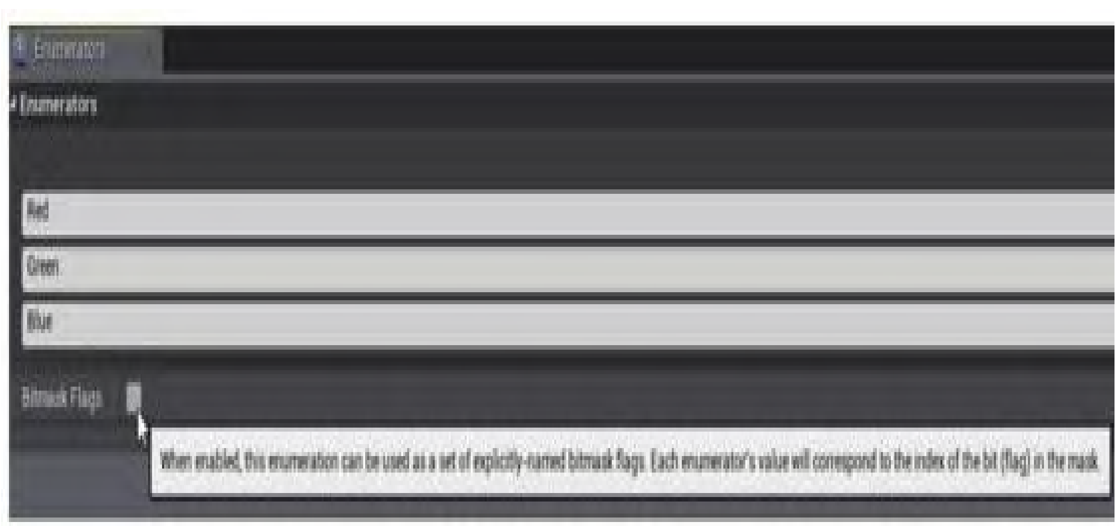



图8-1

**Step 2** 可在蓝图中创建一个整数变量，然后在Details面板中启用Bitmask属性，如图8-2所示。

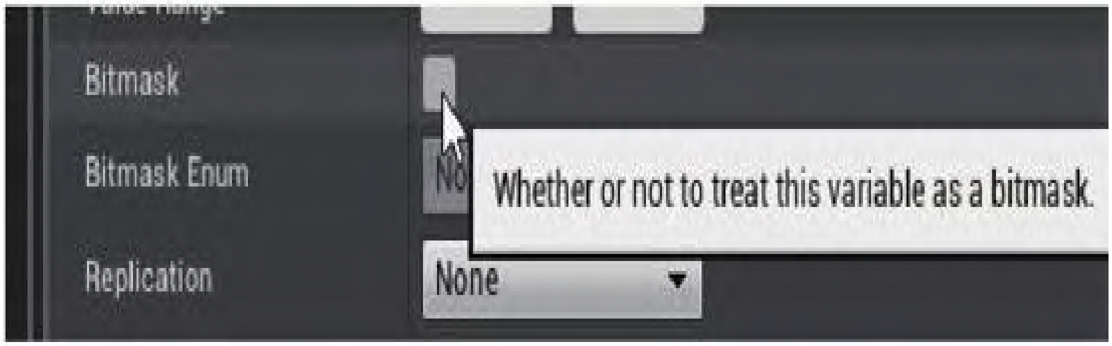



图8-2

这等同于C++代码中的Bitmask元数据。

**Step 3** 在Details面板的Bitmask Enum下拉菜单中选择蓝图枚举（或现成的位标记枚举），如图8-3所示。

**Step 4** 在变量的Set节点上，可使用下拉菜单选择变量值。编译蓝图后，也可在Details面板中设置默认变量值，如图8-4所示。

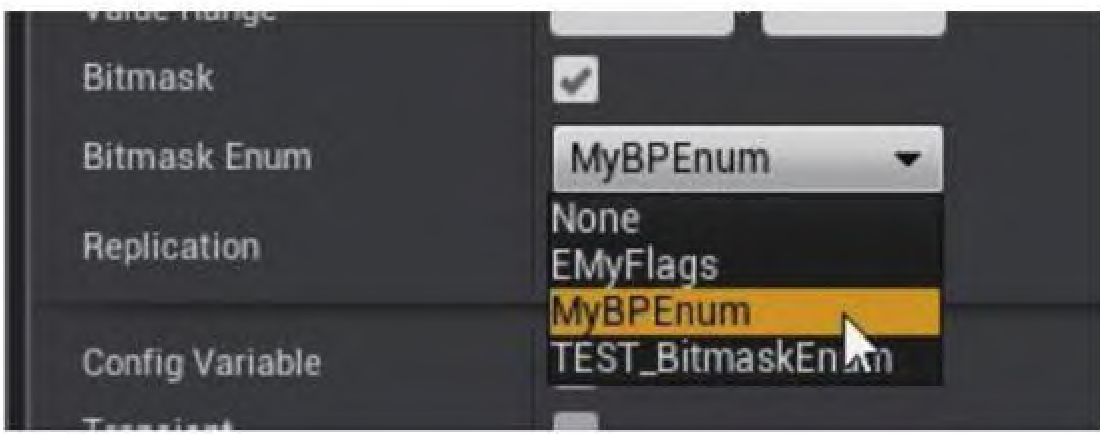



图8-3

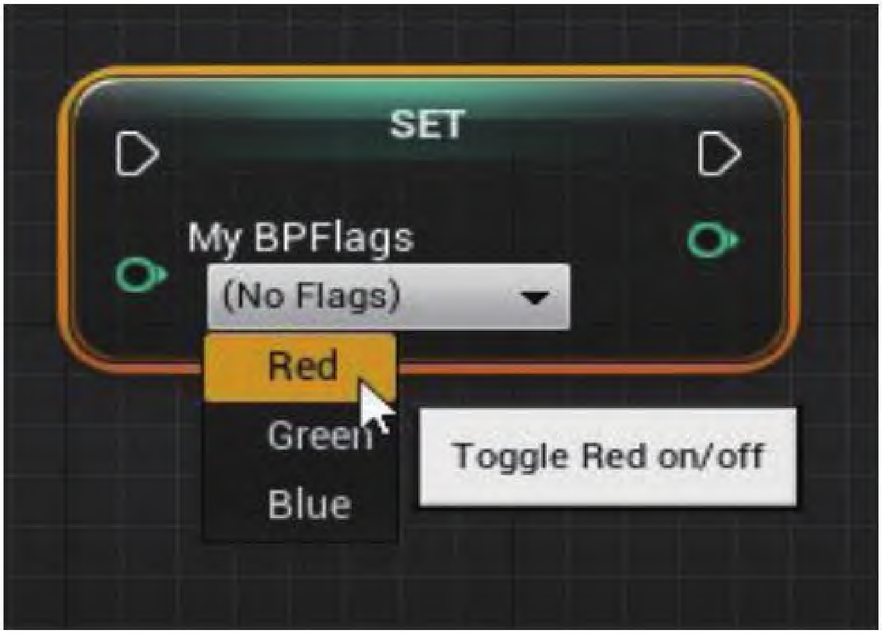



图8-4

8.2

# 在运行时构建位掩码

可通过Make Bitmask节点在蓝图类中创建位掩码文本。

**Step 1** 在图表中右击，然后在快捷菜单中输入“Make Bitmask”。

**Step 2** 选择Make Bitmask，如图8-5所示。

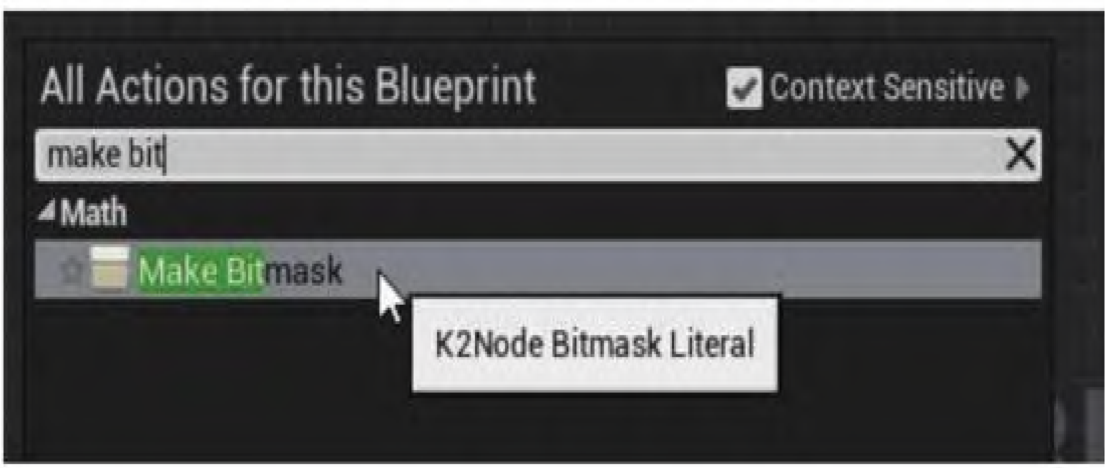



图8-5

**Step 3** 单击节点，然后查看Details面板。使用Bitmask Enum下拉菜单变更与位掩码相关的枚举类型。正如类中定义的整数变量一样，只有启用Bitmask Flags属性的枚举类型被列入下拉菜单，如图8-6所示。

**Step 4** 现在即可在节点上选择Bitmask进行创建，如图8-7所示。

Bitmask Options
Bitmask Enum
EMyFlags
None
EMyFlags
MyBPEnum
TEST_BitmaskEnum

图8-6

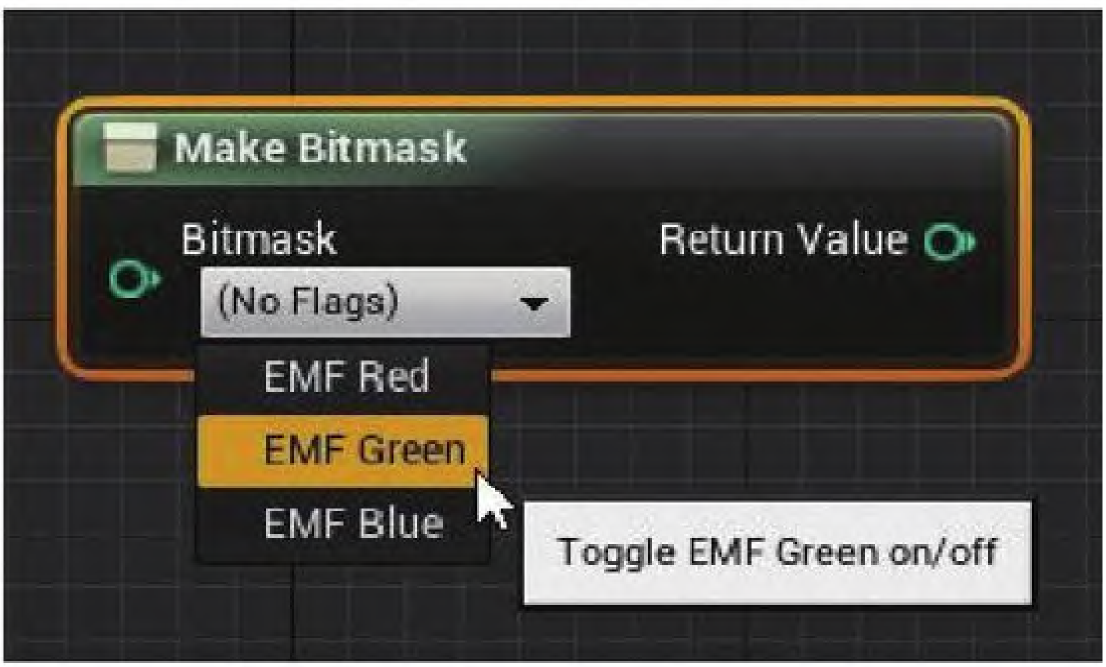

Make Bitmask
Bitmask
Return Value
(No Flags)
EMF Red
EMF Green
EMF Blue
Toggle EMF Green on/off

图8-7

8.3

## 位掩码用法

位掩码可视为蓝图图表中的一个整数值。位掩码整数输出引脚可与接受整数值的任意输入引脚连接；这通常为按位OR、AND、XOR和NOT操作。

在第一个实例中，单个Make Bitmask节点被传递到Bitwise XOR节点，以开关保存在Target Color Flags中的Red标记。存储颜色值的枚举也启用了Bitmask Flags属性，如图8-8所示。

图8-8

在第二个实例中，Select节点、Bitwise NOT节点和Bitwise AND节点全部接受整数作为输入，因此它们可用于在三套颜色中进行选择、开关，然后将结果保存到Target Color Flags位掩码，如图8-9所示。

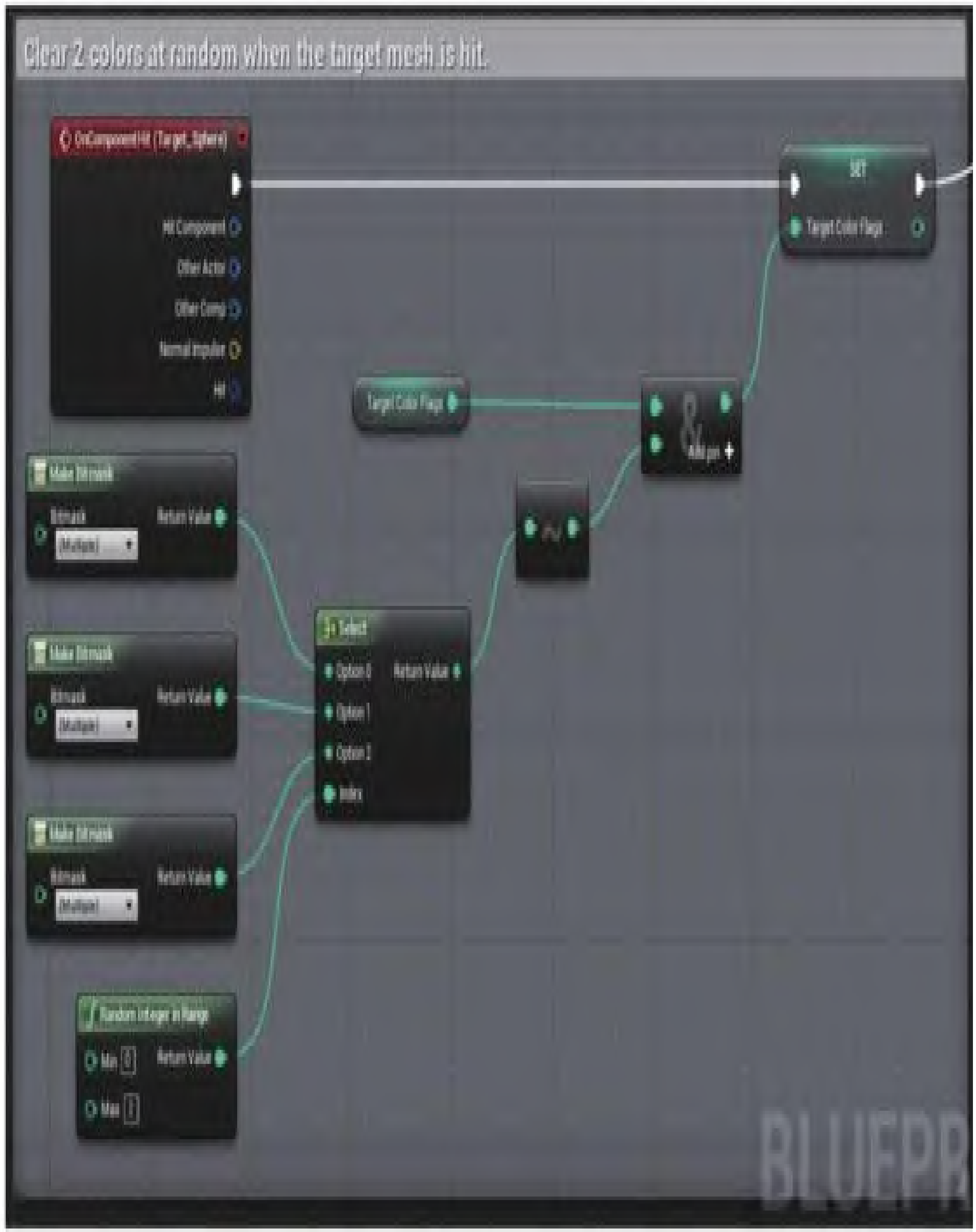

Clear 2 colors at random when the target mesh is hit.
SET
Target Color Flags
Make Bitmask
Bitmask
Return Value
Select
Option 0
Option 1
Option 2
Index
Random Integer in Range
Min
Max
BLUEPR

图8-9

第9章

# 蓝图应用实战

9.1

# 蓝图数组

## 9.1.1　概述

和变量值一样，蓝图也可以在数组中存储数据。如果你不熟悉编程术语，你可以把数组想象成为存在于一个单元中的一组变量，如图9-1所示。

图9-1

数组仅能存放一种类型的值，比如，布尔型数组仅可以存放布尔值。

数组变量包含一个3x3的带颜色网格，表明它们是数组，不是正常的变量。在没有连接的数组中，其网格的中心是黑的。一旦连接后，整个网格将可见，如图9-2所示。

图9-2

## 9.1.2　创建数组

创建数组非常简单，仅需在创建变量时单击Array（数组）网格图标即可，如图9-3所示。

一旦选择了该图标，新建的项就是一个数组，而不是标准的变量，如图9-4所示。

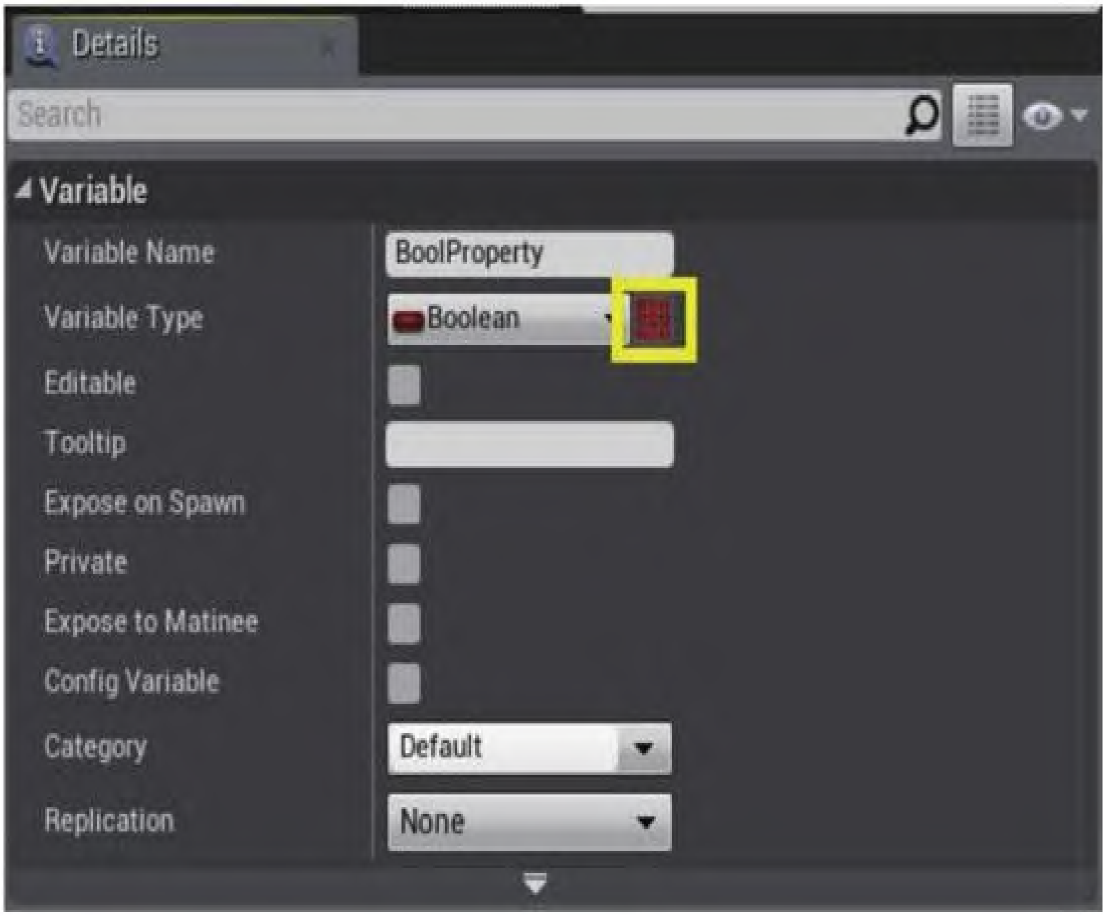

Details
Search
Variable
Variable Name
BoolProperty
Variable Type
Boolean
Editable
Tooltip
Expose on Spawn
Private
Expose to Matinee
Config Variable
Category
Default
Replication
None

图9-3

图9-4

## 9.1.3　编辑数组

你可以通过蓝图默认设置或者沿着蓝图节点网络的任何点来编辑数组的值。这些网络可以存在于构建脚本、函数、宏或事件图表中。

## 9.1.4　数组默认值

设置数组的默认值非常简单。可以简单地创建必要的数组，进入Blueprint Defaults（蓝图默认值）选卡或者蓝图编辑器的Defaults（默认值）模式，你将看到一个以你的数组命名的部分，如图9-5所示。

图9-5

如果在蓝图默认值中你没有看到数组，请确保在创建数组之后你已经编译了你的蓝图。

要想编辑数组的默认值的值：

**Step 1** 在Blueprint Defaults（蓝图默认值）选卡中，单击 按钮。这将会创建一个新的索引。可以重复多次执行这个操作，直到满足了你想让数组包含的元素数为止，如图9-6所示。

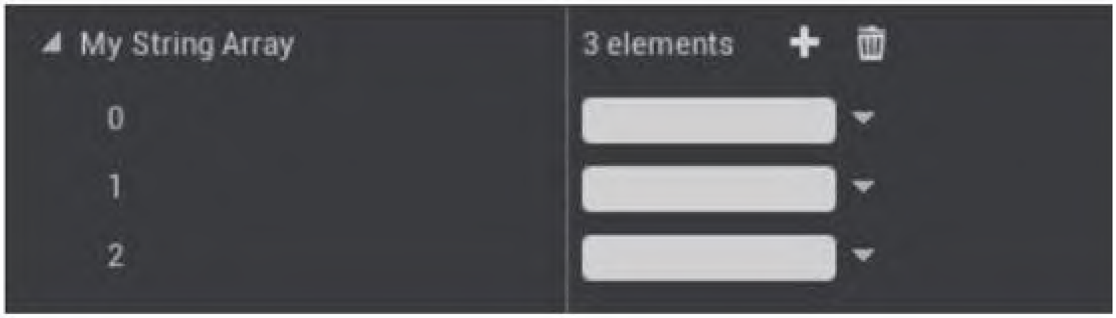



图9-6

在这个示例中，已经添加了三个索引值。

**Step 2**　适当地设置每个值。注意你设置值的方式是由所使用的数组类型决定的，如图9-7所示。

**Step 3**　要想插入、删除或复制一个数组索引，请单击元素项旁边的 按钮来调出编辑菜单，如图9-8所示。

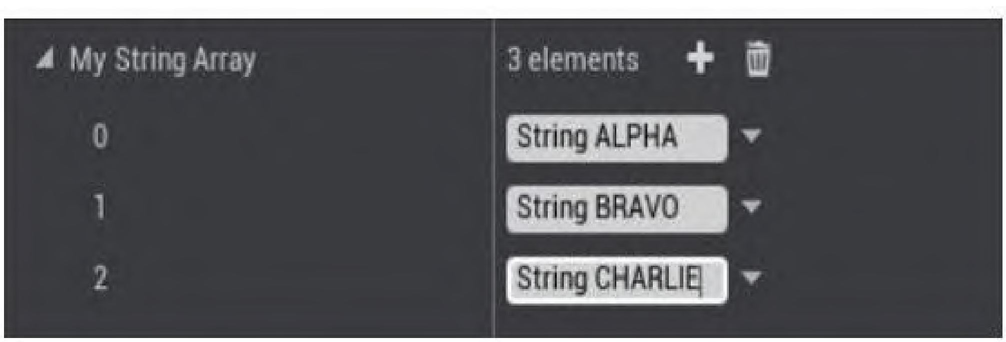



图9-7

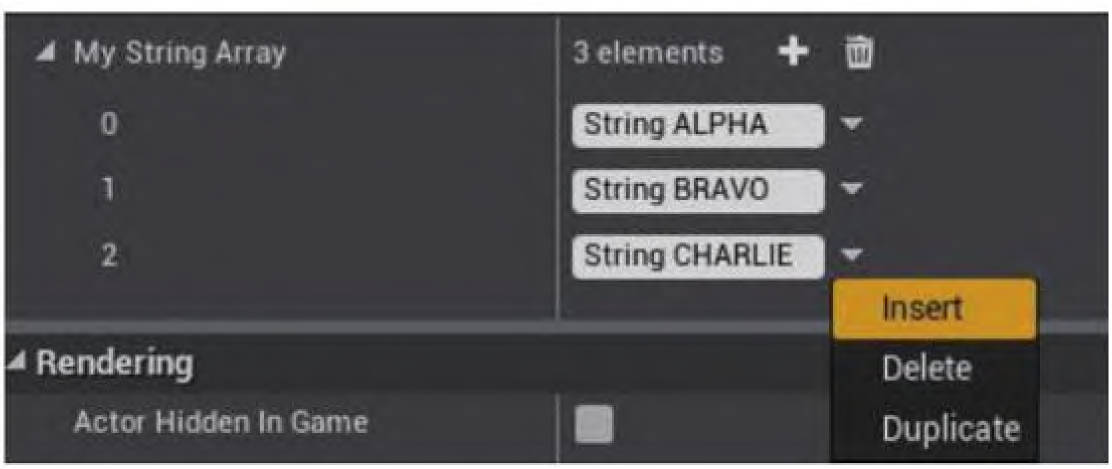



图9-8

如果你从列表的中间添加或删除索引，那么其他的编号将自动地更新。

## 9.1.5 通过节点网络设置数组值

如果你的数组要在运行时进行赋值，那么你一般不需要使用默认值。此时，你可在构建脚本或事件图表中使用节点来填充每个索引。比如，你可以使用Add或Insert节点来添加一个新值到下一个可用索引处，或者插入一个值到给定索引处，如图9-9所示。

图9-9

要想获得蓝图中数组控制节点的完整列表，请参照数组节点页面。

## 9.1.6 数组索引

同时需要注意的一点是，数组中的值的索引值是从0开始的，这意味着第一个索引是0而不是1。比如，如果你添加了5个元素项到数组中，在内部，列表将如下表所示。

| | |
|---|---|
| 索引0 | 第一个元素项 |
| 索引1 | 第二个元素项 |
| 索引2 | 第三个元素项 |
| 索引3 | 第四个元素项 |
| 索引4 | 第五个元素项 |

记住这点非常重要，因为有时候你需要通过其索引值来访问指定的元素项。

## 9.1.7　数组节点

**1. Add Item（添加项目）**

Add Item（添加项目）取入一个数组和一个变量。它将该变量插入数组的尾部，并相应地增加数组的大小，如图9-10所示。

**2. Clear（清除）**

Clear节点将清除相连接的数组中的所有数据，重置数组，并删除数组中的所有索引值，如图9-11所示。

图9-10

图9-11

### 3.Contains Item（包含项目）

Contains Item（包含项目）节点允许搜索一个数组来查找特定项目。如果找到了该项目，那么该节点则输出True，否则会输出False。它不会返回找到该项目所处的索引值，如图9-12所示。

图9-12

### 4.Filter Array（过滤数组）

Filter Array（过滤数组）节点取入一个类型为Actor（或其任何子项）的数组，并基于输入的Class（类）过滤该数组。所得结果是一个新数组，仅包含原始数组中和相连的类的类型相匹配的项目，如图9-13所示。

图9-13

## 5.Find Item（过滤项目）

Find Item（过滤项目）节点取入一个数组和一个变量，然后查找在数组中第一次找到那个变量的值时该值所处的索引编号，如图9-14所示。

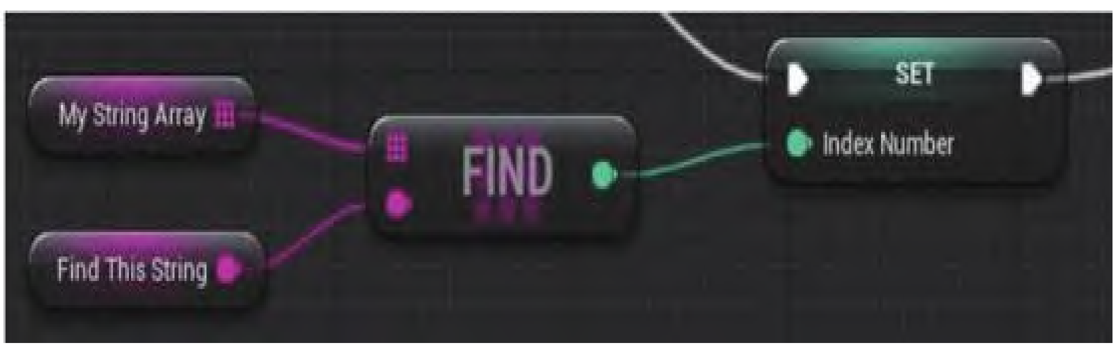



图9-14

## 6.Get（获取）

Get（获取）节点取入一个数组和一个整型值，该整型值用作为索引编号。然后，它返回在索引编号处找到的数组值，如图9-15所示。

图9-15

### 7. Insert Item（插入项目）

Insert Item（插入项目）节点取入一个数组、一个变量值和一个整型值（索引编号）。然后，它把该变量的值插入数组中该索引值指定的位置处，如图9-16所示。

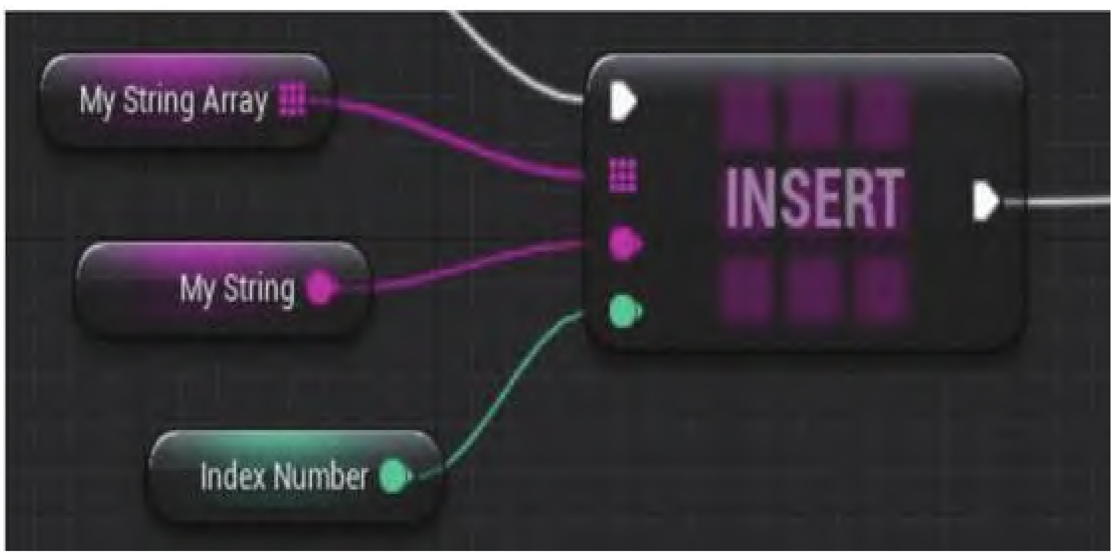



图9-16

### 8. Last Index（最后一个索引值）

Last Index（最后一个索引值）是节点返回数组中最后一个值的索引编号，如图9-17所示。

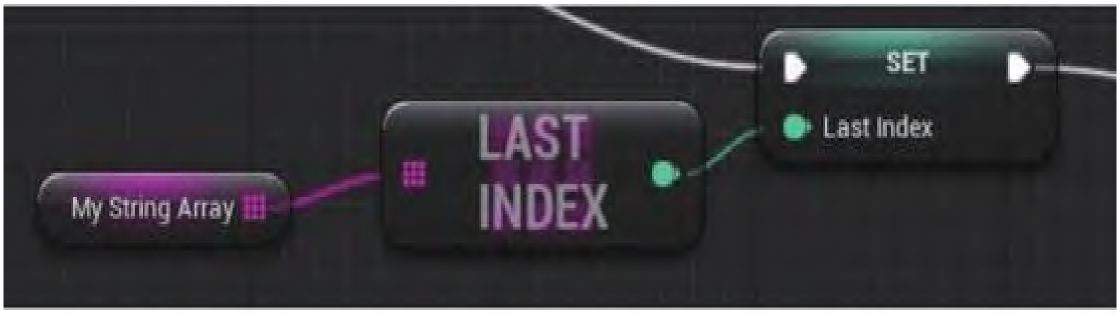



图9-17

### 9. Length（长度）

Length（长度）是返回数组的大小或数组中元素项的个数。如果数组中有N个对象，那么Length将返回N，而数组的有效索引值是从0到N-1，如图9-18所示。

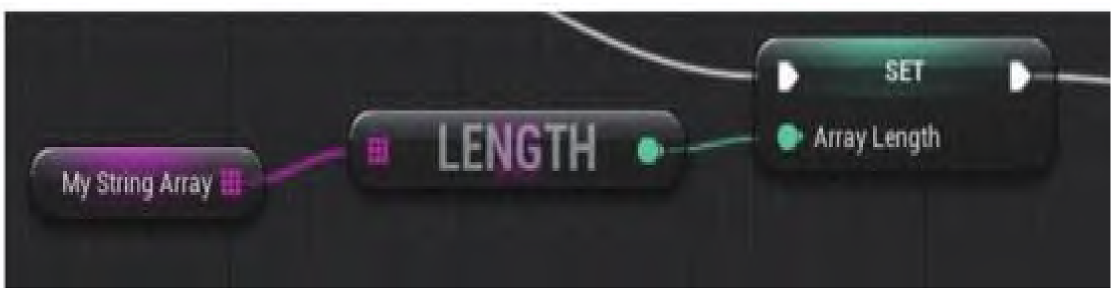



图9-18

### 10. Remove Index（删除索引）

Remove Index（删除索引）是节点通过使用项目的索引编号来从数组中删除那个项目。所有其他的索引编号将自动更新来弥补该空缺，如图9-19所示。

图9-19

11.Remove Item（删除项目）

Remove Item（删除项目）将通过取入一个变量值来从数组中删除一个项目。如果在数组中找到了那个值，则删除它。所有其他的索引编号将自动更新来弥补该空缺。这个节点具有布尔值输出，如果找到了该项目并将其删除了，则返回True，如图9-20所示。

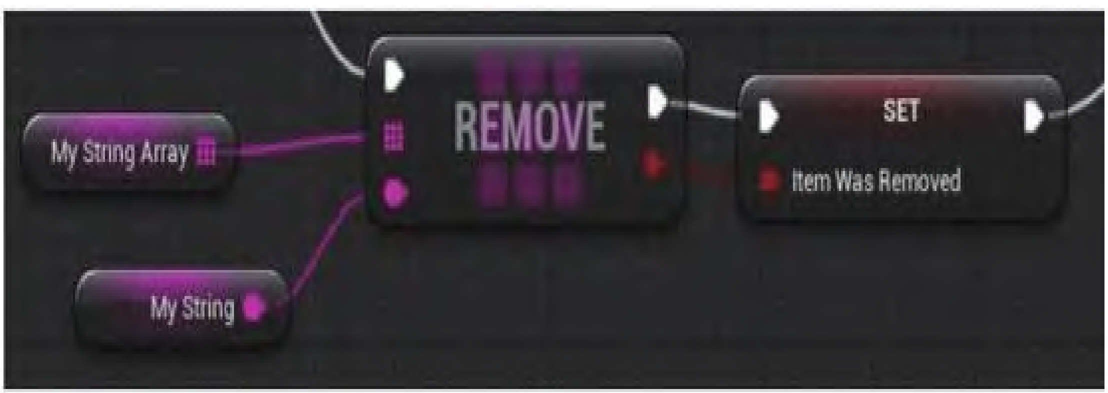



图9-20

12.Resize（调整大小）

Resize（调整大小）节点将取入一个数组和一个整型值，该整型值为该数组的新的容量大小。如果调整大小后的数组小于原始数组，那么所有不适合调整后的数组的元素项都将会丢弃。如果调整大小后的数组大于原始数组，则会在数组尾部添加空元素项，如图9-21所示。

13.Set Array Elem（设置数组元素）

Array Elem（数组元素）允许你将一个数组的一个特定索引设置为特定的值，如图9-22所示。

图9-21

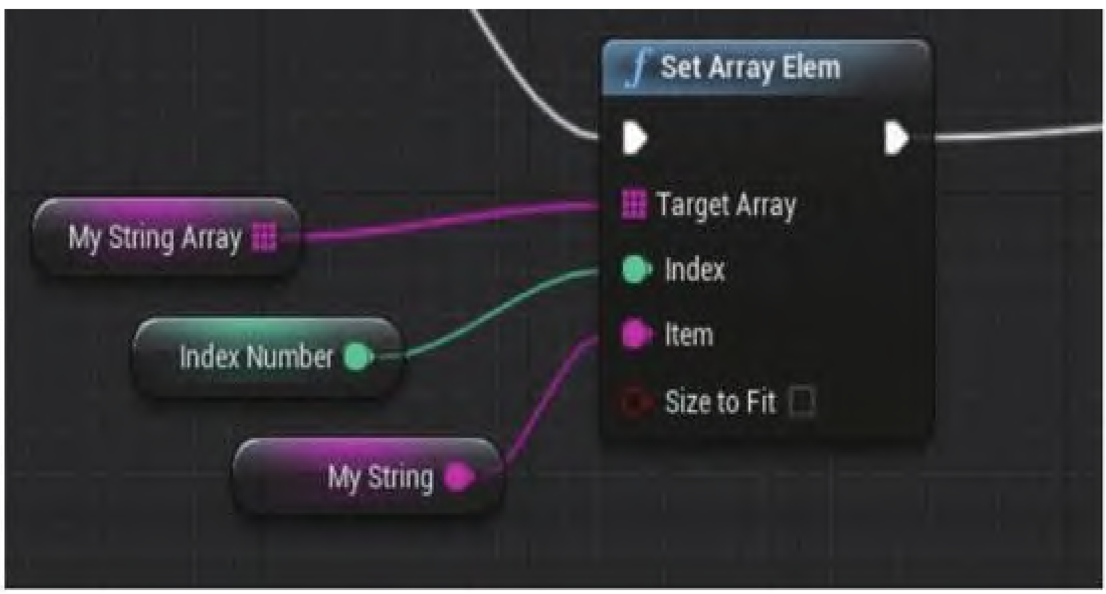



图9-22

9.2

# 流程控制

## 9.2.1　概述

Flow Control（流程控制）使得能在蓝图中清楚地控制执行的流程。此种控制可以多种形式进行，基于某些值为真的情况来选择图表的某个分支来执行，多次执行某个特定分支，以特定顺序执行多个分支，等等。默认的流程控制操作包括了分支（if语句）、循环（for和while语句）、门以及序列。

## 9.2.2　开关节点

开关节点读取数据输入，并会基于该输入值来从匹配的（或可选的默认）执行输出中发送执行流程。可用的开关有以下几种类型：Int（整型）、String（字符串型）、Name（名称型），以及Enum（枚举型）。

一般而言，开关节点会根据其估算的数据类型拥有执行输入以及数据输入。输出均为执行输出。Enum开关会自动从Enum属性中生成输出执行引脚，而Int，String及Name开关拥有可自定义的输出执行引脚。

## 9.2.3　编辑开关节点

当Int、String、或Name开关节点被添加到蓝图时，唯一可用的输出执行引脚为Default（默认）引脚。如输入未能匹配定义的任意其它指定输出引脚，则Default（默认）输出执行引脚将被触发。我们可以通过在引脚上右击并选择Remove Execution Pin（移除执行引脚）或通过对开关节点的Details（细节）选项卡取消勾选Has Default Pin（拥有默认引脚）选项来实现对它的移除，如图9-23所示。

图9-23

编辑Int类型的开关：

**Step 1** 选择Graph（图表）选项卡的开关节点从而在Details（细节）选项卡中打开其属性。

**Step 2** 变更Start Index（开始索引）为你想要比对的最低整数值，如图9-24所示。

**Step 3** 单击开关节点的Add Pin（添加引脚）来对你的Start Index（开始索引）值添加引脚，如图9-25所示。

图9-24

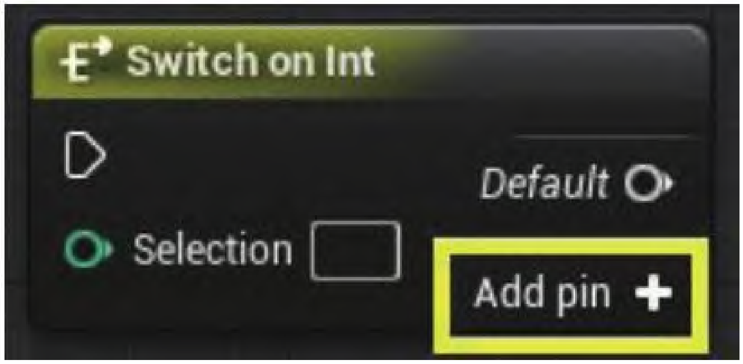



图9-25

进一步单击Add Pin（添加节点）将会添加更多引脚，每次对引脚数量值加1。如需删除执行引脚，可对引脚右击并选择Remove Execution Pin（移除执行引脚）。请注意如移除Switch on Int节点的一个输出引脚，将会导致任意拥有更高值的引脚的值降低1，从而来填充间隔。

编辑Name或String类型的开关：

**Step 1**　选择Graph（图表）选项卡的开关节点从而在Details（细节）选项卡中打开其属性。

**Step 2**　单击Pin Names（引脚名称）右侧的 按钮。

**Step 3**　在出现的文本框中输入你想要String或Name类型的值与其进行比对的值，如图9-26和图9-27所示。

图9-26

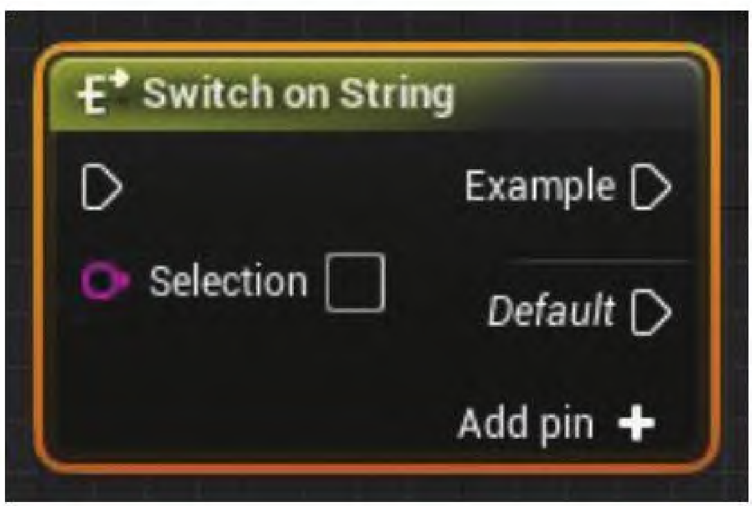



图9-27

对其他你想要添加的引脚重复该过程。引脚也可通过在Details（细节）选项卡中单击 按钮来复制，插入或删除。如需删除执行引脚，你也可以对引脚右击并选择Remove Execution Pin（移除执行引脚）。

## 9.2.4　标准流程控制节点

标准流程控制节点提供了一系列方法来控制执行的流程，如图9-28所示。

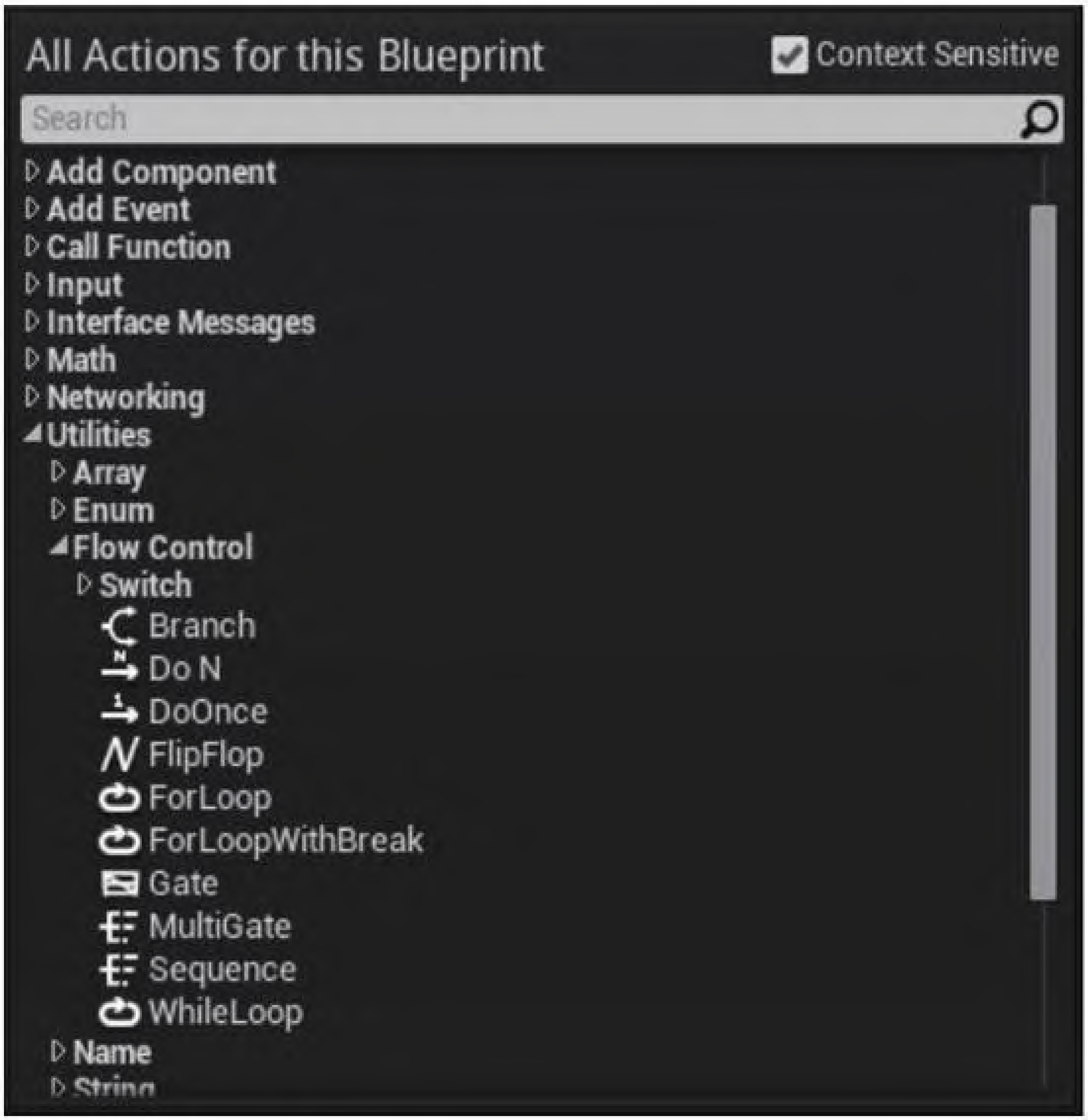



图9-28

#### 9.2.4.1 Branch

在面对单个True/False判定的情况下，Branch（分支）节点是一种创建基于判断的流程的简单方式。在执行后，分支节点会查找附加的布尔变量的输入值，并在合适的输出节点下方来输出执行脉冲值，如图9-30所示。

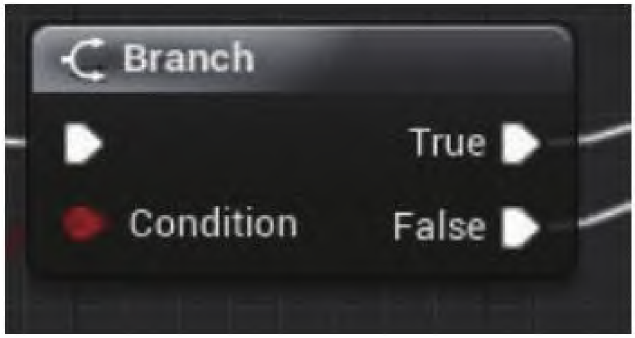



图9-29

在下图这个简单的示例中，分支节点会查找布尔变量的当前状态。如该值为true，则其会将光照的颜色设置为红色。如其为false，则设置为蓝色，见图9-30和下表。

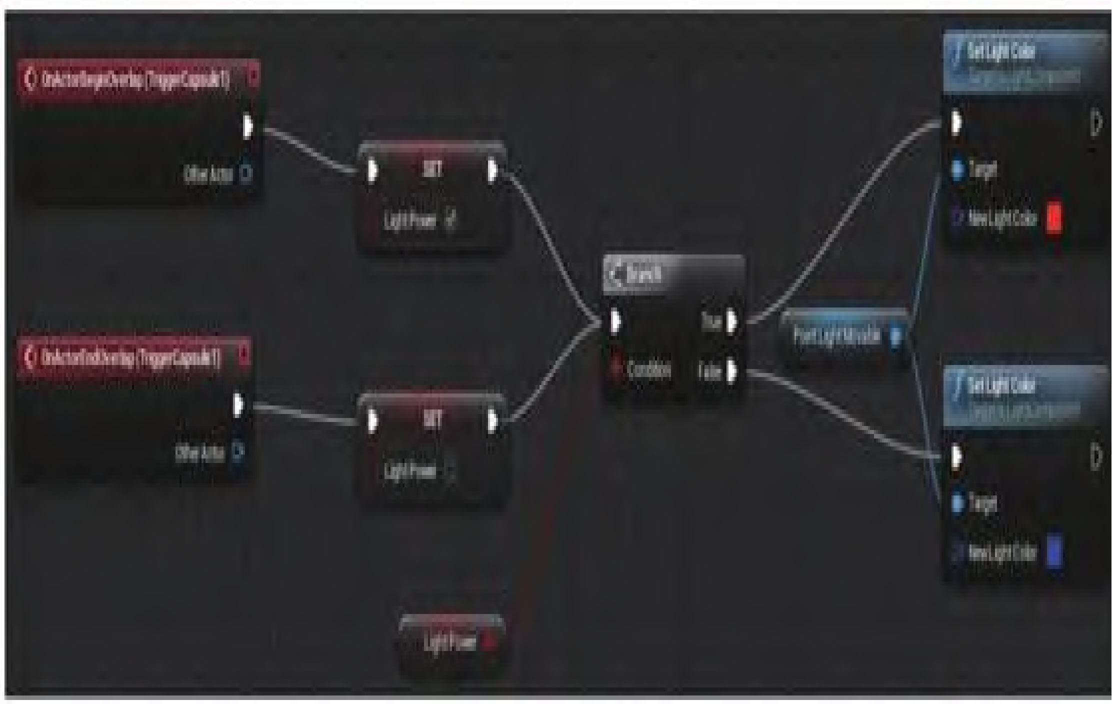



图9-30

| 项目 | 描述 |
| --- | --- |
| 输入引脚 | |
| (Unlabeled) | 该执行输入会触发分支检查 |
| Condition | 取入用来显示哪个输出引脚将被触发的布尔值 |
| 输出引脚 | |
| True | 如输入的状态为 true，则输出执行脉冲 |
| False | 如输入的状态为 false，则输出执行脉冲 |

#### 9.2.4.2 DoN

DoN节点将N次触发执行脉冲。在达到限制后，它将停止所有的输出执行，直到脉冲被传入其Reset（重置）输入，如图9-31所示。

图9-31

例如，你可以启动车辆20次，然后在绑定到Reset（重置）输入的加油事件被激活前，无法再次启动车辆，见图9-32和下表。

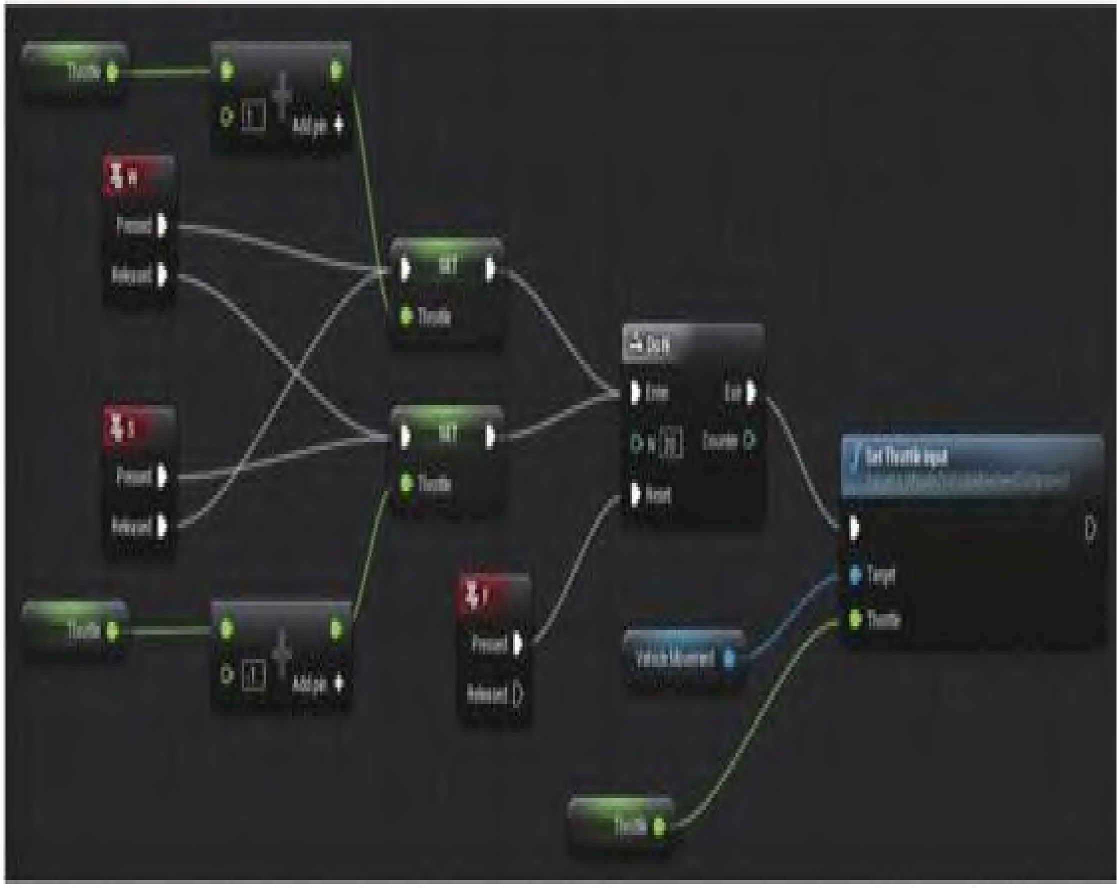


图9-32

| 项　目 | 描　述 |
| --- | --- |
| 输入引脚 | |
| Enter | 该执行输入会触发 DoN 检查 |
| n | 该输入设置了 DoN 节点将会触发的次数 |
| Reset | 该执行输入将会重置 DoN 节点，这样它可以被再次触发 |
| 输出引脚 | |
| Exit | 该执行引脚仅在 DoN 的触发次数尚未达到 N 次时，才会被触发，或者在其 Reset（重置）输入被调用时进行触发 |

### 9.2.4.3　DoOnce

DoOnce节点，正如名称所显示的，将会仅仅触发执行脉冲一次。在之后，它将停止所有的输出执行，直到脉冲被传入其Reset（重置）输入。该节点等同于DoN节点中N=1的情况，如图9-33所示。

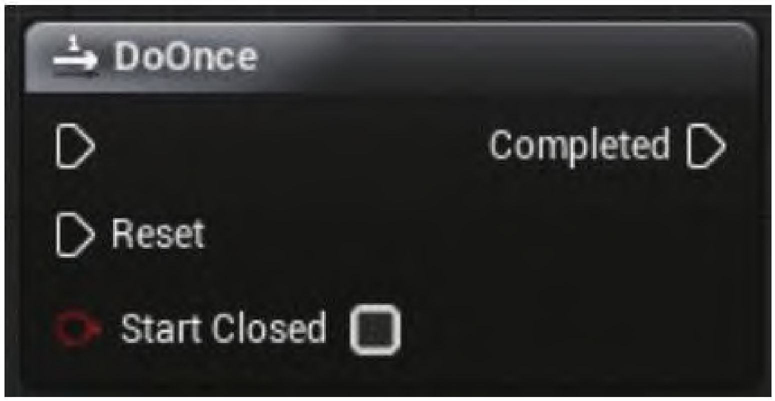



图9-33

我们举例来说，你可以对一扇开启的门的节点网络设置DoOnce，这样这扇门将仅仅开启一次。不过你可以绑定一个触发事件到Reset（重置）输入，这样会导致在触发器被激活时，门会再次打开，见图9-34和下表。

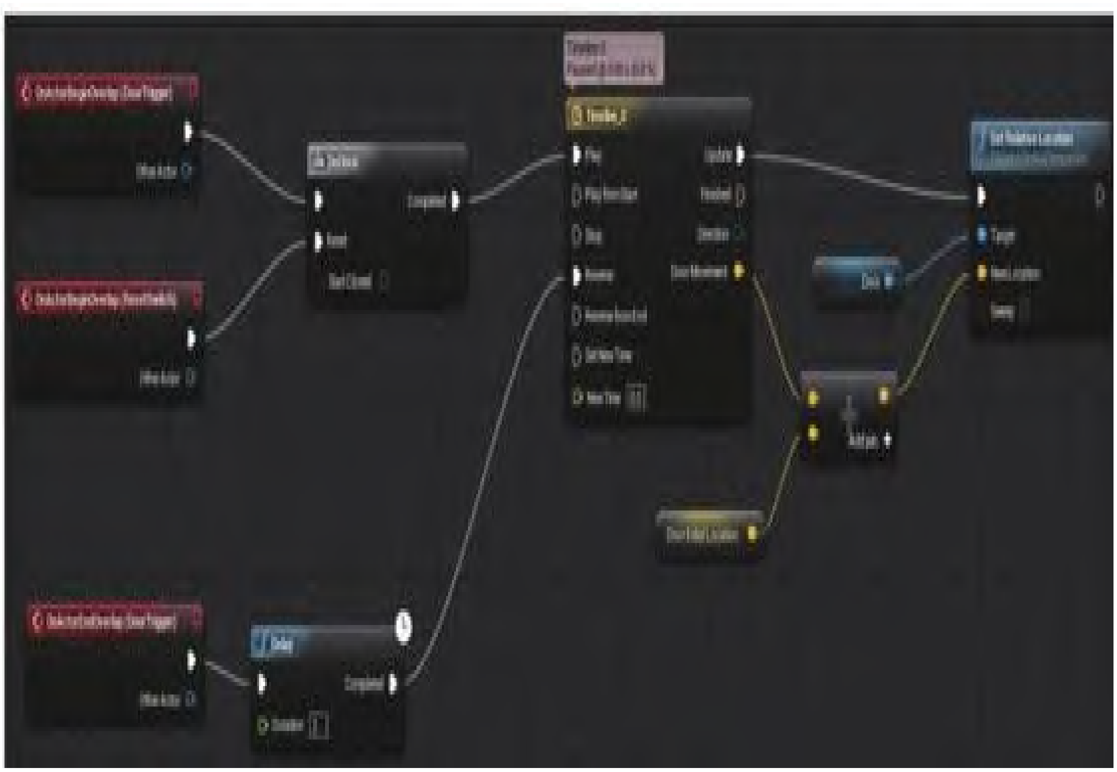


图9-34

| 项目 | 描述 |
|---|---|
| 输入引脚 | |
| （Unlabeled） | 该执行输入会触发 DoOnce 检查 |
| Reset | 该执行输入将会重置 DoOnce 节点，这样它可以被再次触发 |
| 输出引脚 | |
| Completed | 该执行引脚仅在 DoN 节点尚未被触发时，才会被触发，或者在其 Reset（重置）输入被调用时进行触发 |

### 9.2.4.4 FlipFlop

FlipFlop节点取入执行输出并在两个执行输出间切换。其第一次被调用时，将输出A。第二次被调用时，将输出B，然后再是A，然后又是B，循环往复。该节点同时有布尔变量输出，使你可以追溯输出A何时被调用，见图9-35和图9-36以及下表。

图9-35

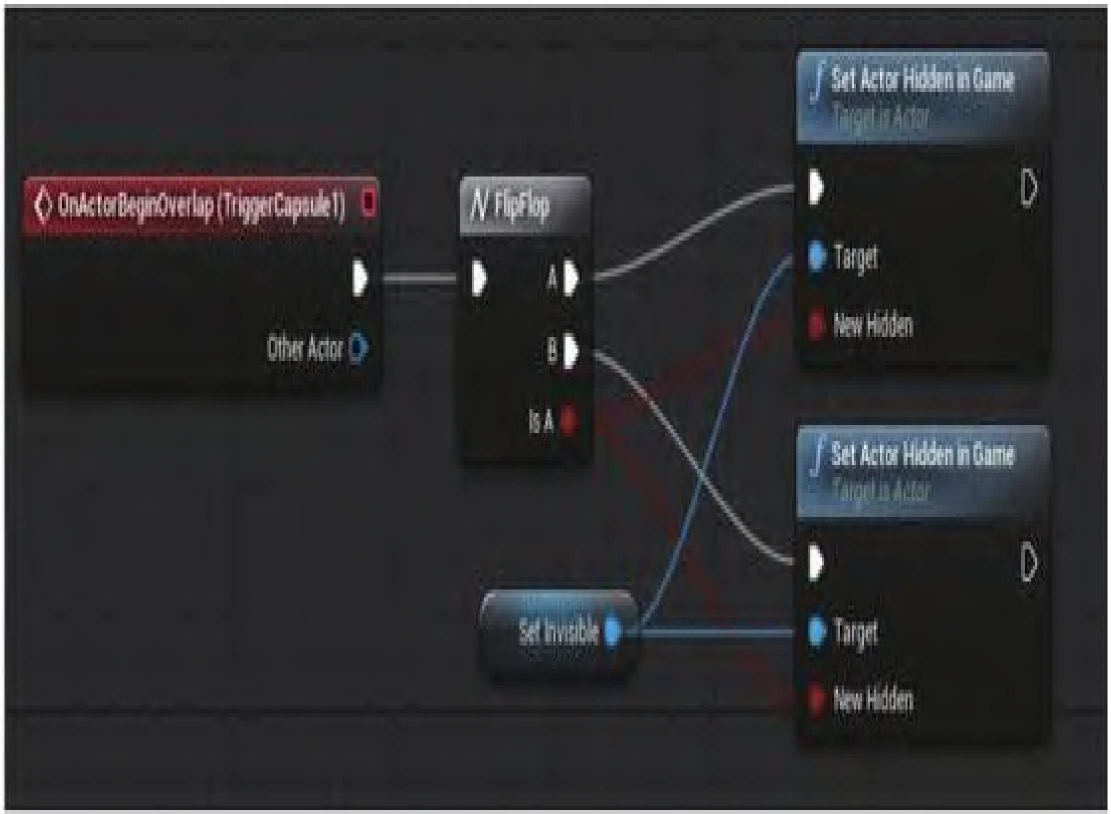



图9-36

| 项　目 | 描　述 |
|---|---|
| 输入引脚 | |
| (Unlabeled) | 该执行输入会触发 FlipFlop |
| 输出引脚 | |
| A | 该输出引脚在首次及之后 FlipFlop 被触发的每个奇数次被调用 |
| B | 该输出引脚在第二次及之后 FlipFlop 被触发的每个偶数次被调用 |
| Is A | 输出布尔变量值，以表明输出 A 是否被触发。这个函数生效后，将会在每次 FlipFlop 节点被触发后，在 true 和 false 间切换 |

#### 9.2.4.5 ForLoop

ForLoop节点的工作原理等同于标准的代码循环，将在开始和结束之间的每个索引触发执行脉冲，如图9-37所示。

图9-37

在此简单示例中，当玩家触碰简单的关卡触发器时，将触发循环。该循环将迭代10次，每次都会调用Print String（输出字符串），它会记录前缀信息以及当前迭代信息。

循环迭代将会在不同的帧间发生，这样的话大量循环可能会影响性能表现，见图9-38和下表。

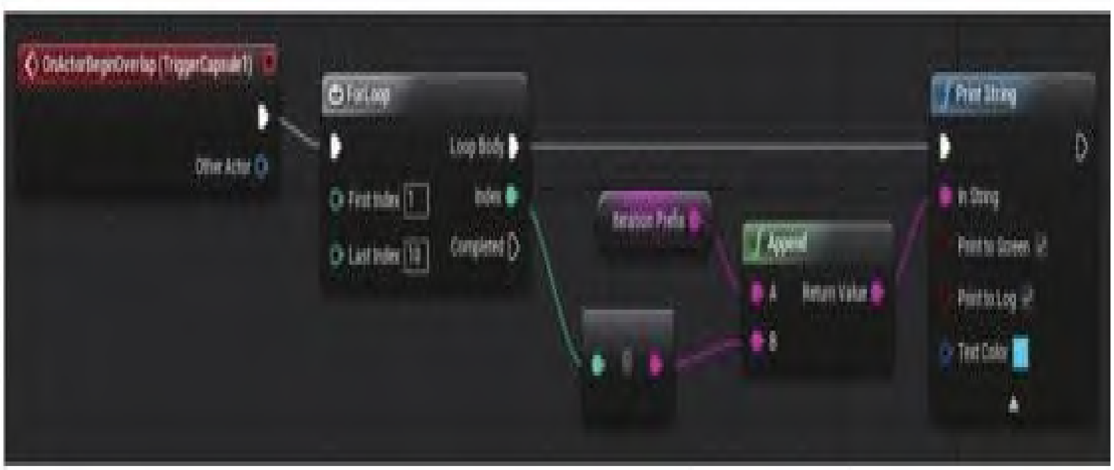



图9-38

| 项　目 | 描　述 |
| --- | --- |
| 输入引脚 | |
| (Unlabeled) | 该执行输入会启动循环 |
| First Index | 取入表示循环首个索引的整数值 |
| Last Index | 取入表示循环最后索引的整数值 |
| 输出引脚 | |
| Loop Body | 当其在不同的索引间移动时，对循环的每次迭代输出执行脉冲 |
| Index | 输出循环的当前索引 |
| Completed | 当循环完成时，触发标准的执行输出引脚 |

### 9.2.4.6　ForLoopWithBreak

ForLoopWithBreak节点包含了能中断循环的输入引脚，除此之外，它运行的方式与ForLoop节点非常相似，如图9-39所示。

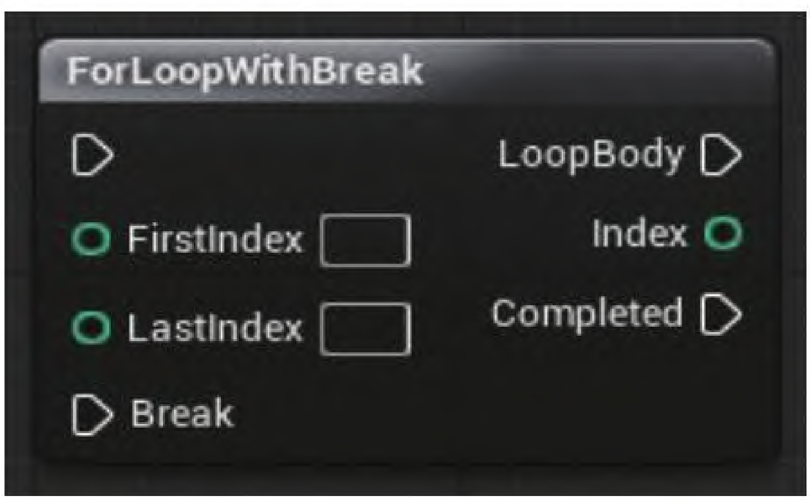



图9-39

在此简单示例中，当玩家触碰简单的关卡触发器时，将触发循环。该循环会迭代1000次，每次都会触发分支，该分支会检查循环是否达到了500次迭代。如果没有达到，屏幕上会出现当前迭代次数的信息。在其超过500次后，分支会调用Custom Event（自定义事件），该值将中断循环。Custom Event（自定义事件）用来让整体更容易看懂，从而不需要把线重新拉回Break（中断）输入引脚附近。

循环迭代将在不同的帧间发生，这样的话大量循环可能会影响性能表现，见图9-40和9-41以及下表。

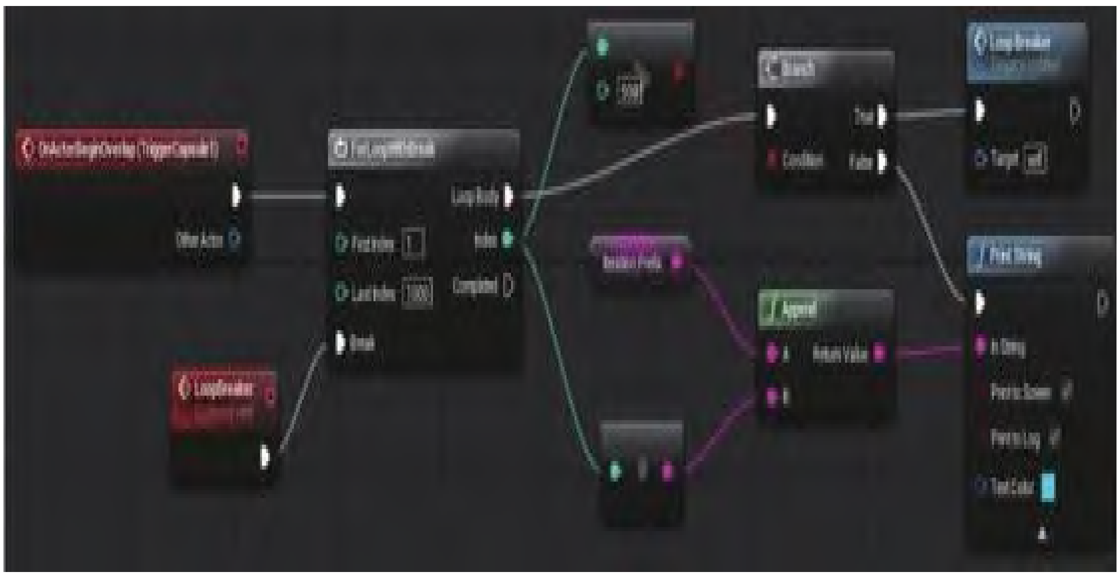


图9-40

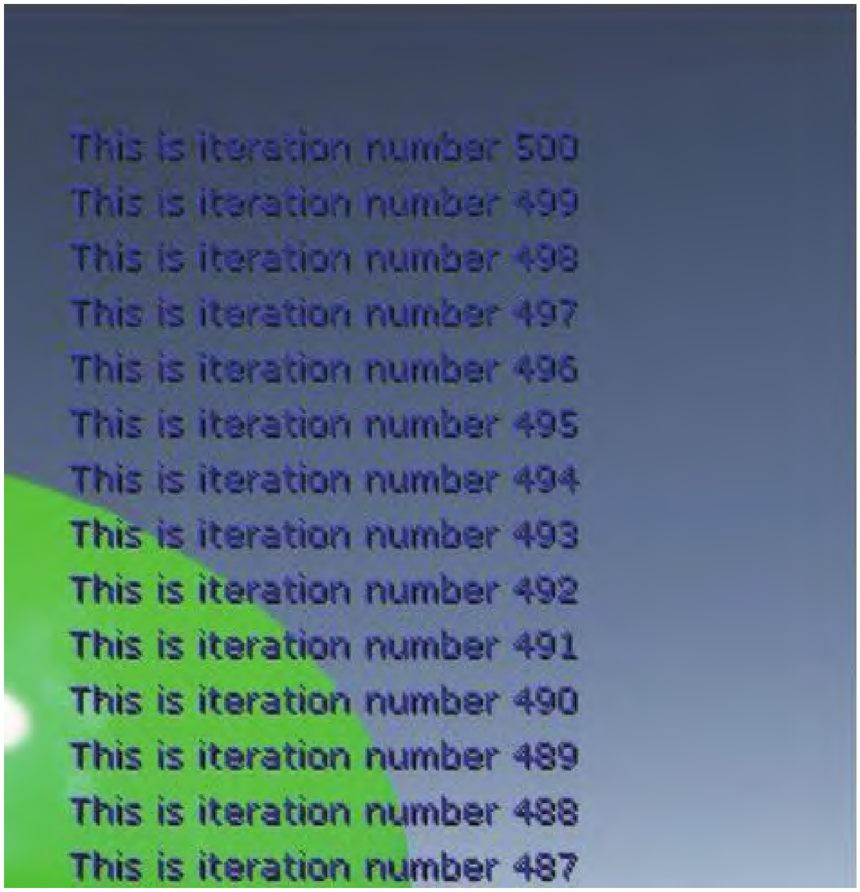

This is iteration number 500
This is iteration number 499
This is iteration number 498
This is iteration number 497
This is iteration number 496
This is iteration number 495
This is iteration number 494
This is iteration number 493
This is iteration number 492
This is iteration number 491
This is iteration number 490
This is iteration number 489
This is iteration number 488
This is iteration number 487

图9-41

| 项　目 | 描　述 |
|---|---|
| 输入引脚 | |
| (Unlabeled) | 该执行输入会启动循环 |
| First Index | 取入表示循环首个索引的整数值 |
| Last Index | 取入表示循环最后索引的整数值 |
| Break | 该执行输入会中断循环 |
| 输出引脚 | |
| Loop Body | 当其在不同的索引间移动时，对循环的每次迭代输出执行脉冲 |
| Index | 输出循环的当前索引 |
| Completed | 当循环完成时，触发标准的执行输出引脚 |

### 9.2.4.7　Gate

Gate（门）节点用来开启和关闭执行流。简单来说，Enter输入取入执行脉冲，同时门的当前状态（开启或关闭）将决定这些脉冲是否从Exit输出中传出，如图9-42所示。

在此简单示例中，一条没有轨迹的时间轴，with both auto-play and looping activated（同时启用了自动播放并激活了循环），将对门节点的Enter输入引脚进行更新。关卡中有两个触发器。一个触发器开门，另一个关门。如门为开启状态，执行脉冲会离开Exit引脚，此时Print String（输出字符串）被调用，它会登记一条信息到屏幕上。当玩家触碰Close（关闭）触发器时，门会关闭并且信息会停止显示。如果随后触碰Open（开启）触发器，该信息会再次出现，见图9-43和下表。

图9-42

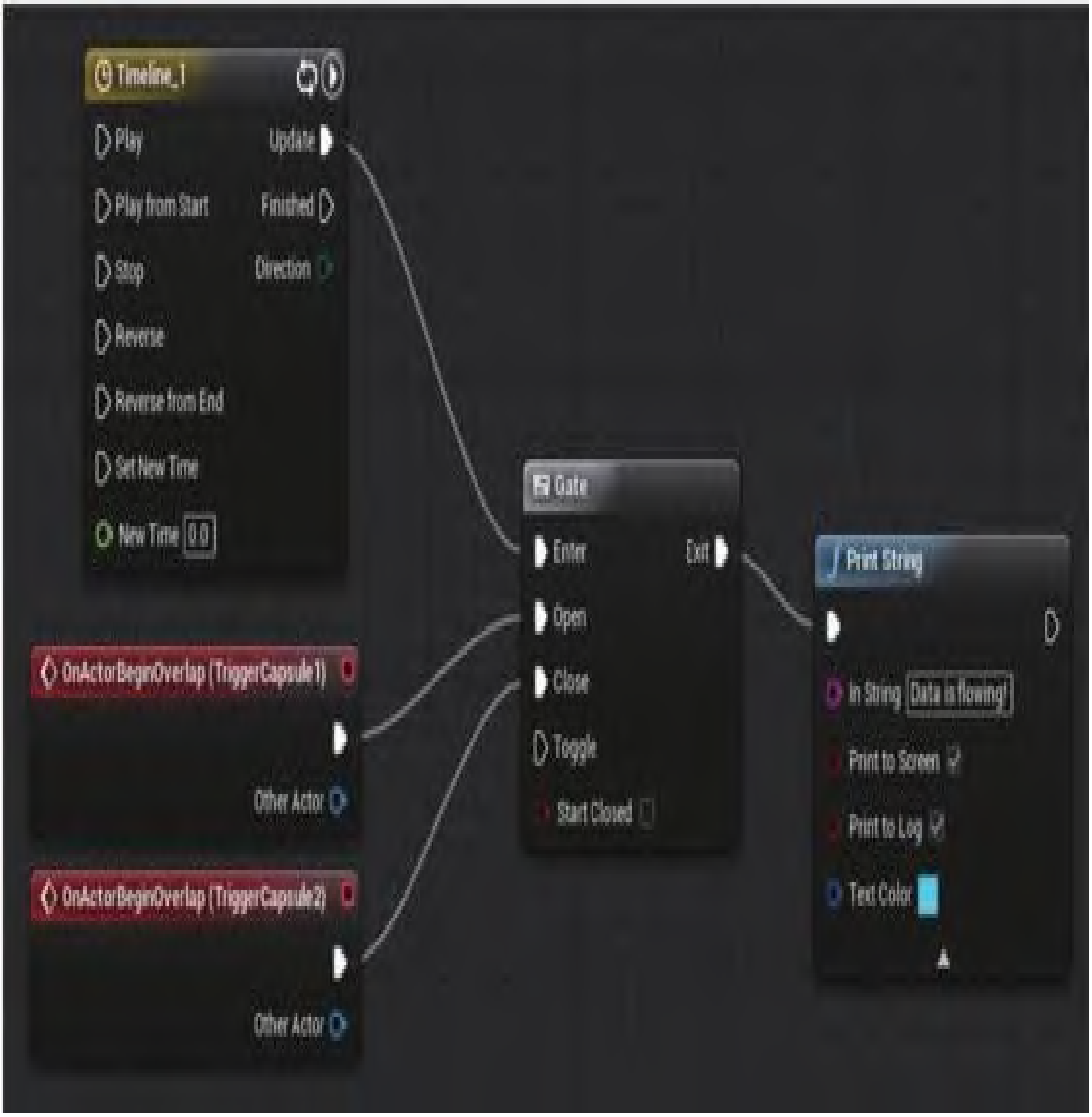

Timeline_1
Play
Update
Play from Start
Finished
Stop
Direction
Reverse
Reverse from End
Set New Time
New Time
0.0
OnActorBeginOverlap (TriggerCapsule1)
Other Actor
OnActorBeginOverlap (TriggerCapsule2)
Other Actor
Gate
Enter
Exit
Open
Close
Toggle
Start Closed
Print String
In String
Data is flowing!
Print to Screen
Print to Log
Text Color

图9-43

| 项　目 | 描　述 |
| --- | --- |
| 输入引脚 | |
| Enter | 此执行输入表示任何由门控制的执行 |
| Open | 此执行引脚设置门的状态为 Open（开启），使执行脉冲传送到 Exit 输出引脚 |
| Close | 此执行引脚设置门的状态为 Closed（关闭），使执行脉冲停止传送到 Exit 输出引脚 |
| Toggle | 此执行引脚反向转换门的当前状态。Open（开启）变成 Closed（关闭），反之亦然 |
| Start Closed | 此布尔变量输入决定了门的起始状态。如设置为 true，则门的初始状态为关闭 |
| 输出引脚 | |
| Exit | 如果门的当前状态为 Open（开启），则任何流入 Enter 输入引脚的执行脉冲将会离开 Exit 输出引脚。如门为 Closed（关闭）状态，则 Exit 引脚将无法产生作用 |

### 9.2.4.8 MultiGate

MultiGate取入单个数据脉冲并将其传送到任意数量的潜在输出。这个过程随机按顺序发生，可能会循环，如图9-44所示。

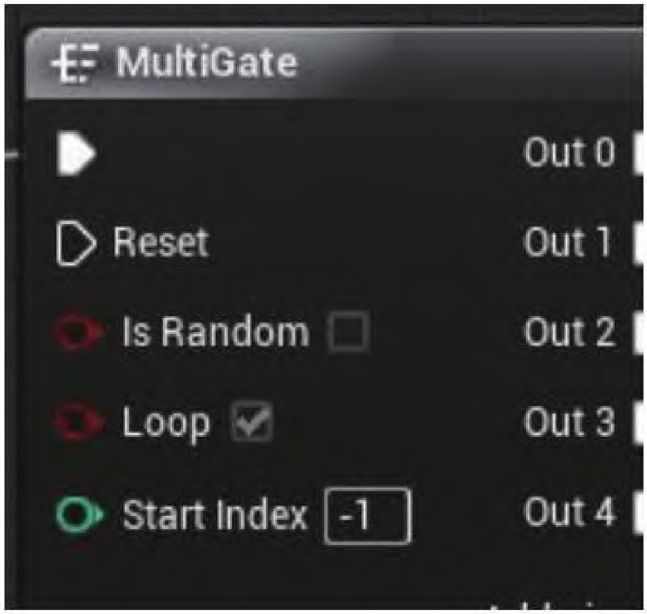



图9-44

在本示例中，一个简单的循环及自动播放的时间轴每隔半秒输出脉冲。该脉冲到达MultiGate并做相应传送，触发一系列Print String（输出字符串）节点中的一个，而此系列节点如按顺序播放，会展示一条特殊信息，见图9-45和下表。

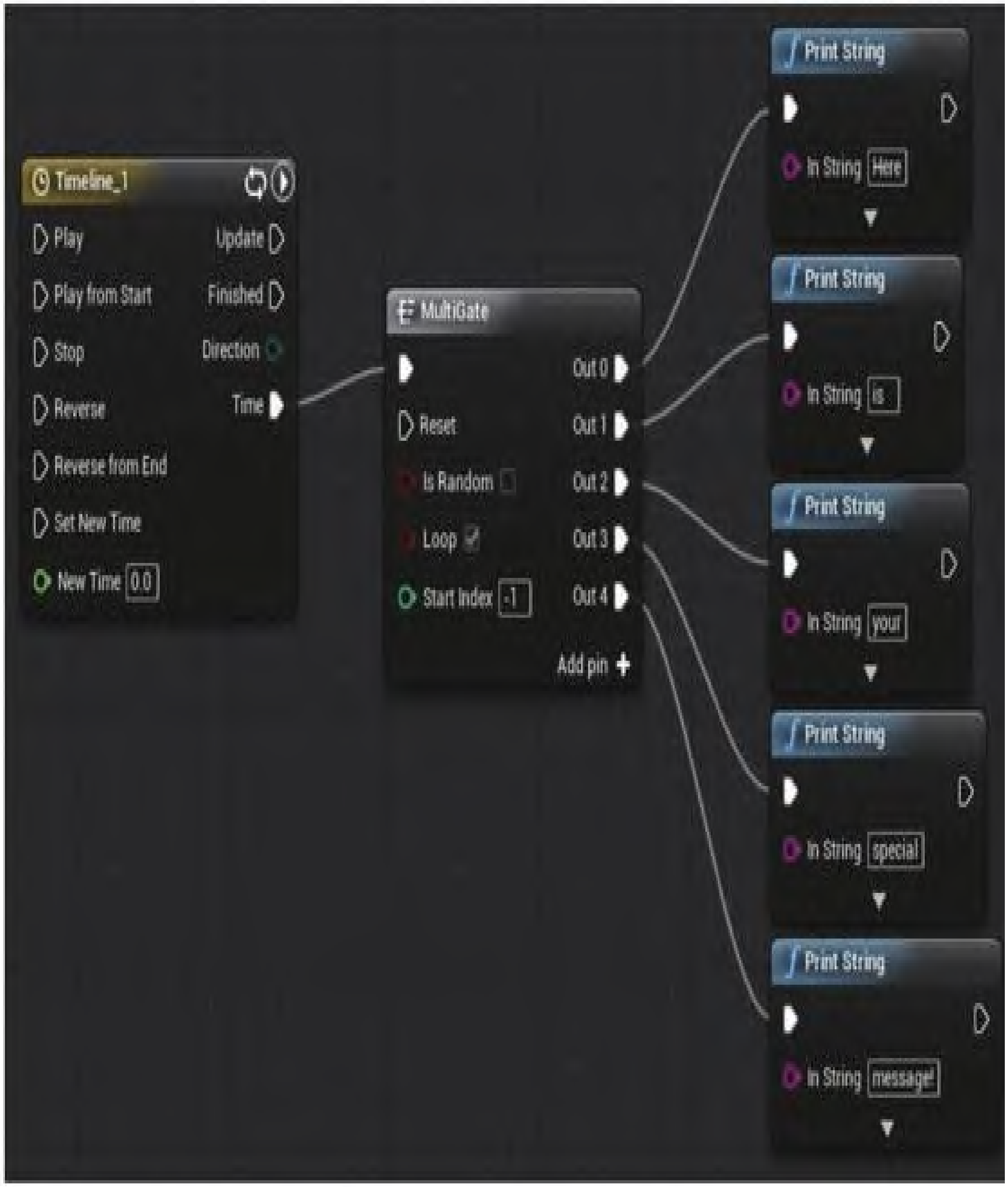

Timeline_1
Play
Update
Play from Start
Finished
Stop
Direction
Reverse
Time
Reverse from End
Set New Time
New Time
0.0
MultiGate
Out 0
Reset
Out 1
Is Random
Out 2
Loop
Out 3
Start Index
-1
Out 4
Add pin
Print String
In String
Here
Print String
In String
is
Print String
In String
your
Print String
In String
special
Print String
In String
message!

图9-45

| 项　目 | 描　述 |
|---|---|
| 输入引脚 | |
| (Unlabeled) | 取入任意需经过此 MultiGate 的脉冲的主输入 |
| Reset | 默认把当前输出索引设置回 0，或如果该值非 -1，则将其设置为当前设置的 Start Index（开始索引）值 |
| Is Random | 如设置为 true，则以随机顺序选择输出 |
| Loop | 如设置为 true，则输出将持续以循环方式重复。如设置为 false，则 MultiGate 值会在使用了所有输出后停止运行 |
| Start Index | 取入一个整数值来表示 MultiGate 应首先使用的输出索引值。-1 在这里表示未指定起始点 |
| 输出引脚 | |
| Out # | 每个输出引脚代表了 MultiGate 节点可用来发送脉冲的可能输出引脚 |
| Add pin | 尽管算不上真正的输出引脚，此按钮使你能够添加你想要的任意数量的输出。如需移除输出引脚，可通过右击并选择 Remove Output Pin（移除输出引脚）来执行 |

### 9.2.4.9　Sequence

Sequence（序列）节点使得单个执行脉冲能按顺序触发一系列事件。节点可能有任意数量的输出，所有的输出引脚都会在序列节点一获得输入时就被调用。它们将总是按顺序被调用，但不会有任何延迟。对一般用户来说，输出引脚看起来好像被同时触发了一样，如图9-46所示。

本例中，序列节点在关卡的起始处被调用。然后它按顺序触发5个Print String（输出字符串）节点。但是，由于没有延迟，这些信息看起来是同时出现的，见图9-47和下表。

图9-46

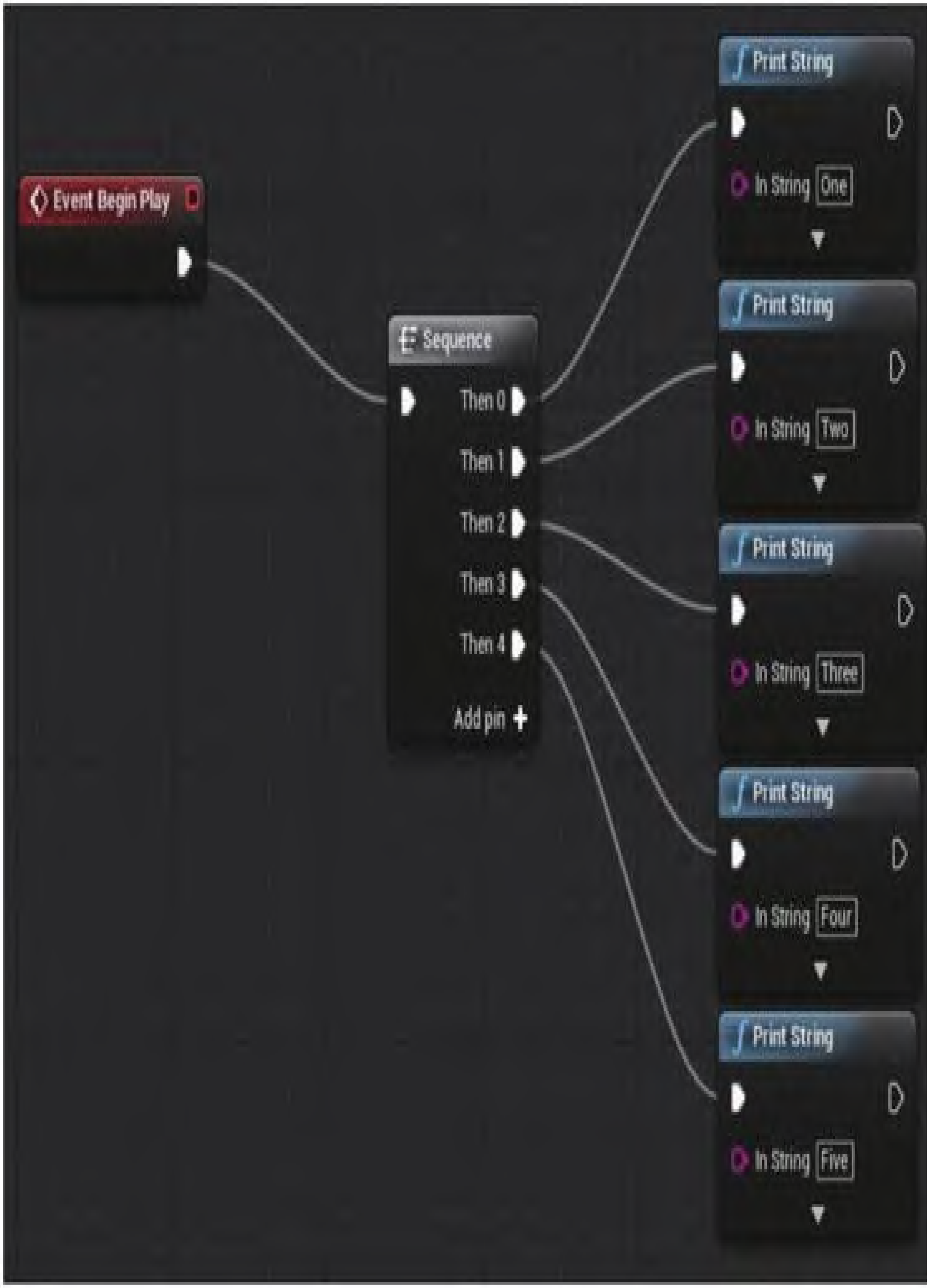

Event Begin Play
Sequence
Then 0
Then 1
Then 2
Then 3
Then 4
Add pin
Print String
In String
One
Print String
In String
Two
Print String
In String
Three
Print String
In String
Four
Print String
In String
Five

图9-47

| 项　目 | 描　述 |
|---|---|
| 输入引脚 | |
| (Unlabeled) | 取入任意需经过此 Sequence(序列)的脉冲的主输入 |
| 输出引脚 | |
| Out # | 每个输出引脚代表了 Sequence(序列)节点可用来发送脉冲的可能输出引脚 |
| Add pin | 尽管算不上真正的输出引脚，此按钮使你能够添加你想要的任意数量的输出。如需移除输出引脚，可通过右击并选择 Remove Output Pin(移除输出引脚)来执行 |

### 9.2.4.10 WhileLoop

只要特定值为true，则WhileLoop节点将输出一个结果。在循环的每个迭代中，它会查看其输入布尔值的当前状态。一旦它读取到false，该循环中断，如图9-48所示。

图9-48

在本例中，WhileLoop与整数型迭代网络一起被使用，从而计算循环已经运行的次数。当它到达第10次迭代时，测试布尔变量被设置为false，从而终止循环的行为。

在对While loops进行编程时，要尤其注意防止其无限循环，见图9-49和下表。

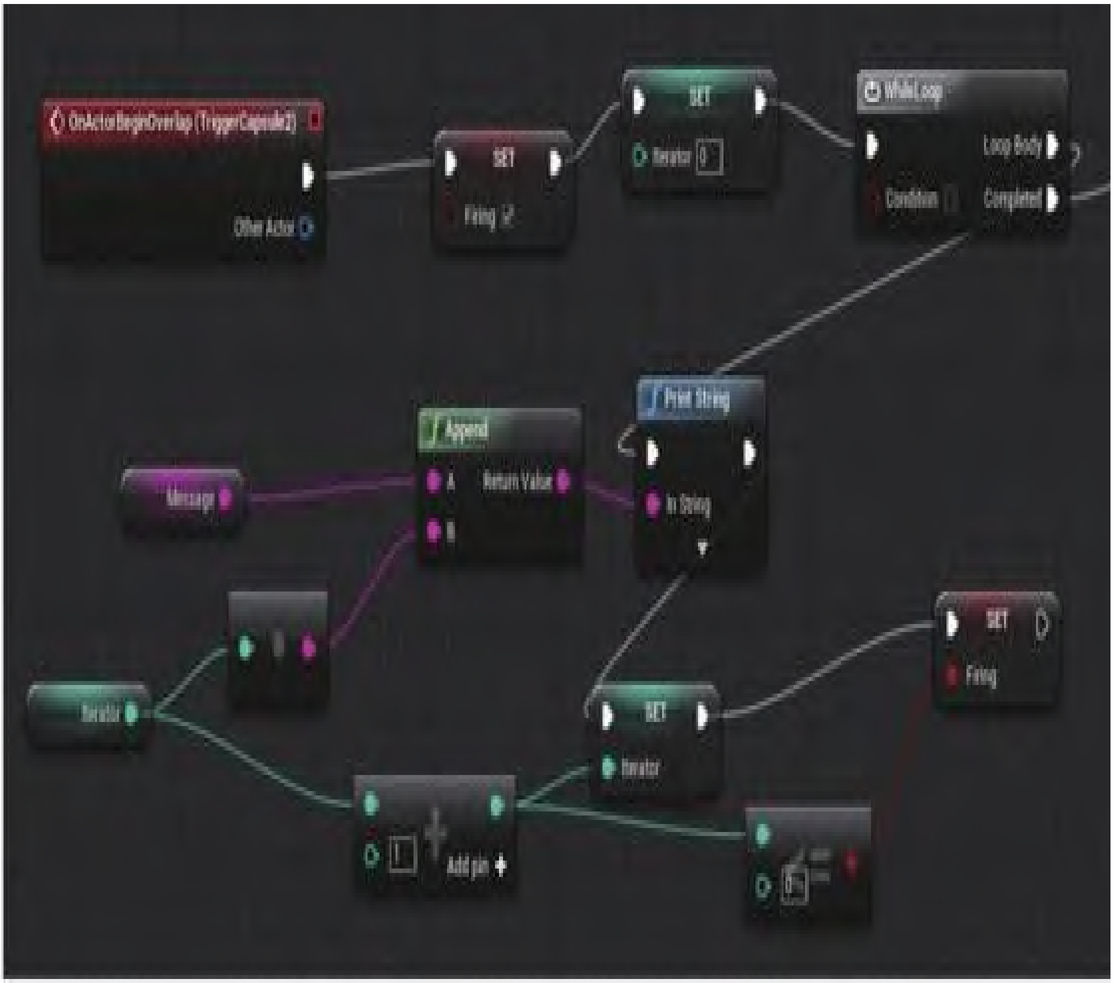



图9-49

| 项　目 | 描　述 |
|---|---|
| 输入引脚 | |
| （Unlabeled） | 取入任意驱动此 WhileLoop 的脉冲的主输入 |
| 输出引脚 | |
| Loop Body | 在不同的索引间移动时，对循环的每次迭代输出执行脉冲 |
| Completed | 当循环一结束，就触发标准的执行输出引脚 |

9.3

# 函数

Functions（函数）是属于特定蓝图的节点图表，该节点图表可以执行或者从蓝图中的另一个图表中进行调用。函数具有一个单一的入口点，由和该函数具有相同名称的一个节点指定，该节点包含了一个单一的执行输出引脚。当从另一个图表调用函数时，会激活该输出执行引脚，促使执行连接网络。

## 9.3.1　访问修饰符

当创建函数时，你可以指定可以访问及不能访问这些函数的其他对象。这可以通过设置Access Specifier（访问修饰符）属性来完成，该属性可以在选中函数的详细信息面板中找到，如图9-50所示。

方式,只有其所在的类可以调用该函数。

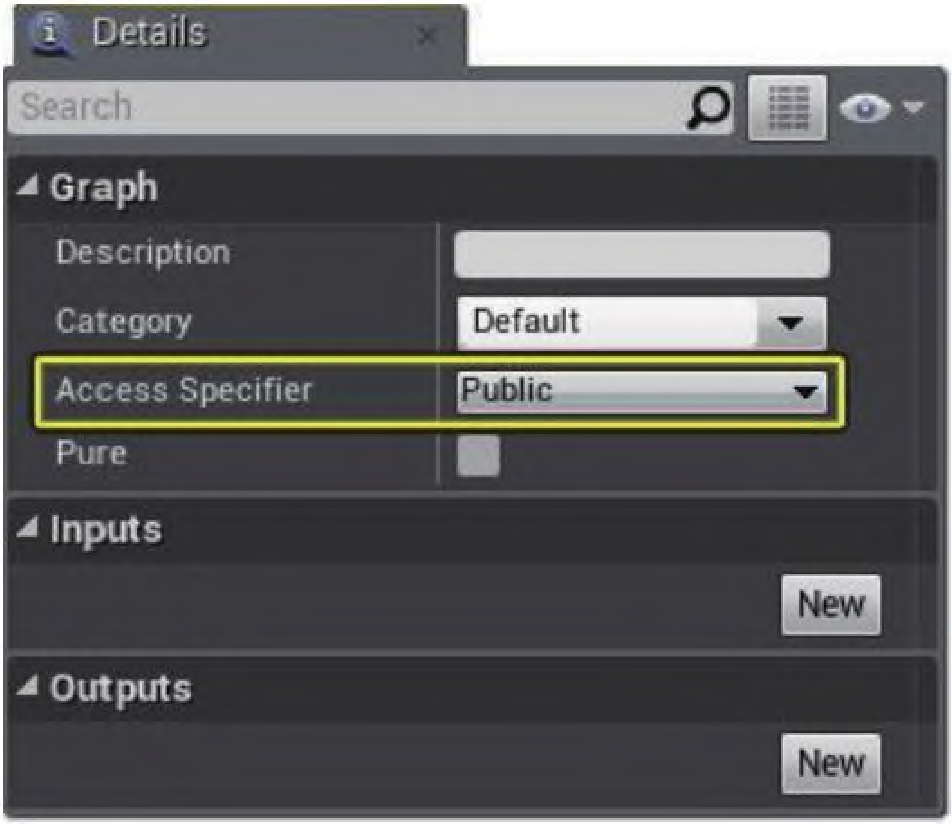



图9-50

该属性包含以下设置:

| 设　置 | 描　述 |
|---|---|
| Public(公有) | 设置函数为public,意味着其他任何对象都可以访问该函数,这是默认设置 |
| Protected(保护) | 设置函数为protected,意味着仅当前蓝图及任何继承当前蓝图的蓝图可以访问该函数 |
| Private(私有) | 设置函数为private,意味着仅能从当前蓝图调用该函数 |

## 9.3.2　纯函数和非纯函数

函数可以是纯函数也可以是非纯函数。主要的区别是纯函数不会以任何方式修改状态或者类的成员,而非纯函数可以自由地修改状态。纯函数一般用于Getter函数或者仅输出一个数据值的操作符。

非纯函数必须通过在事件图表中将执行线连接到函数调用节点来显式地执行。非纯函数通过使用以下这些方式之一进行指定：

•在代码中定义的函数的函数声明中指定BlueprintCallable关键字。

•将通过蓝图编辑器添加的函数的Pure（纯函数）复选框保留为未选中状态。

纯函数连接到数据引脚上，当需要依赖于纯函数的数据时，编译器会自动地执行这些函数。这意味着对于每个连接到纯函数上的节点，都会调用一次纯函数。纯函数通过使用以下这些方式之一进行指定：

•在代码中定义的函数的函数声明中指定BlueprintPure关键字。

•选中通过蓝图编辑器添加的函数的Pure（纯函数）复选框。

### 9.3.3 创建函数

**1.在蓝图中创建函数**

要想在一个类蓝图或关卡蓝图中创建函数：

**Step 1** 在Graph（图表）模式，从My Blueprint（我的蓝图）选卡下，单击New Function（新建函数）按钮，如图9-51所示。

**Step 2** 输入你的函数的名称，如图9-52所示。

将在蓝图编辑器的Graph Editor（图标编辑器）选卡的一个新选卡中打开你的函数。

你也可以在我的蓝图选卡中右击并选择Function（函数）来创建一个函数，如图9-53所示。

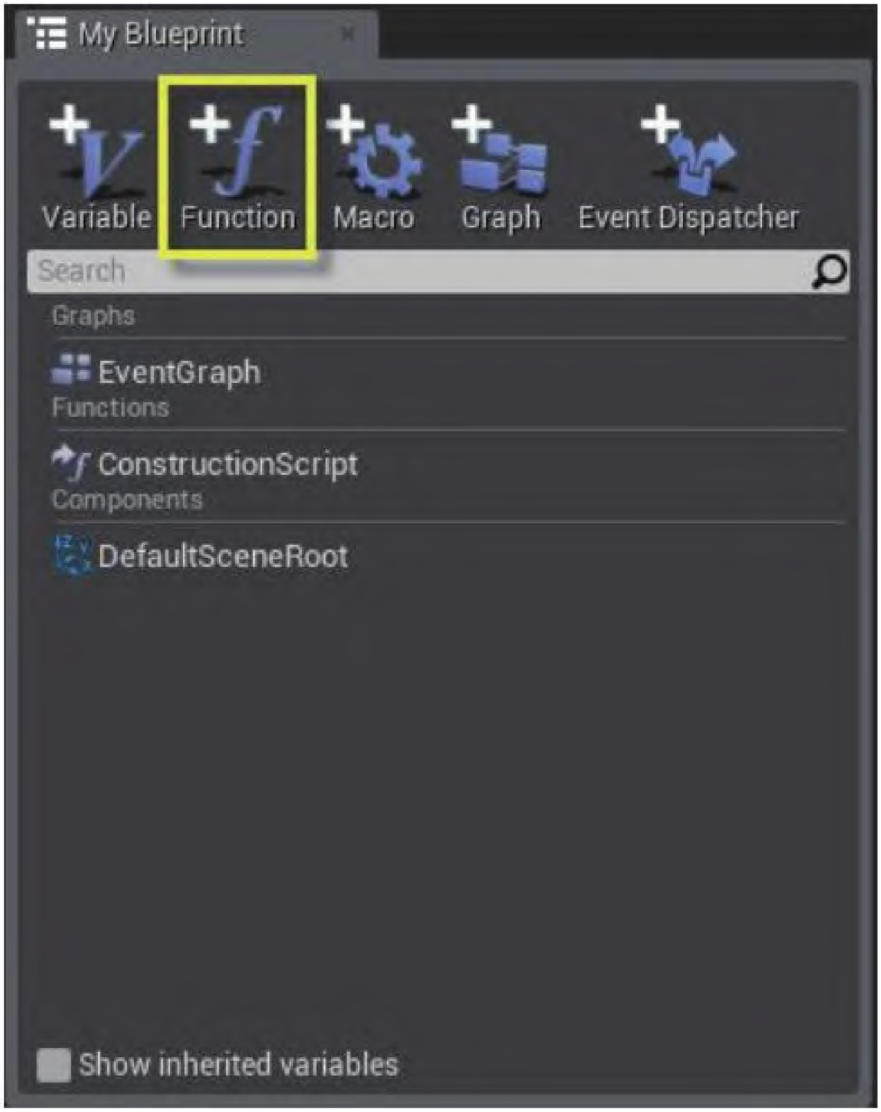

My Blueprint
Variable
Function
Macro
Graph
Event Dispatcher
Search
Graphs
EventGraph
Functions
ConstructionScript
Components
DefaultSceneRoot
Show inherited variables

图9-51

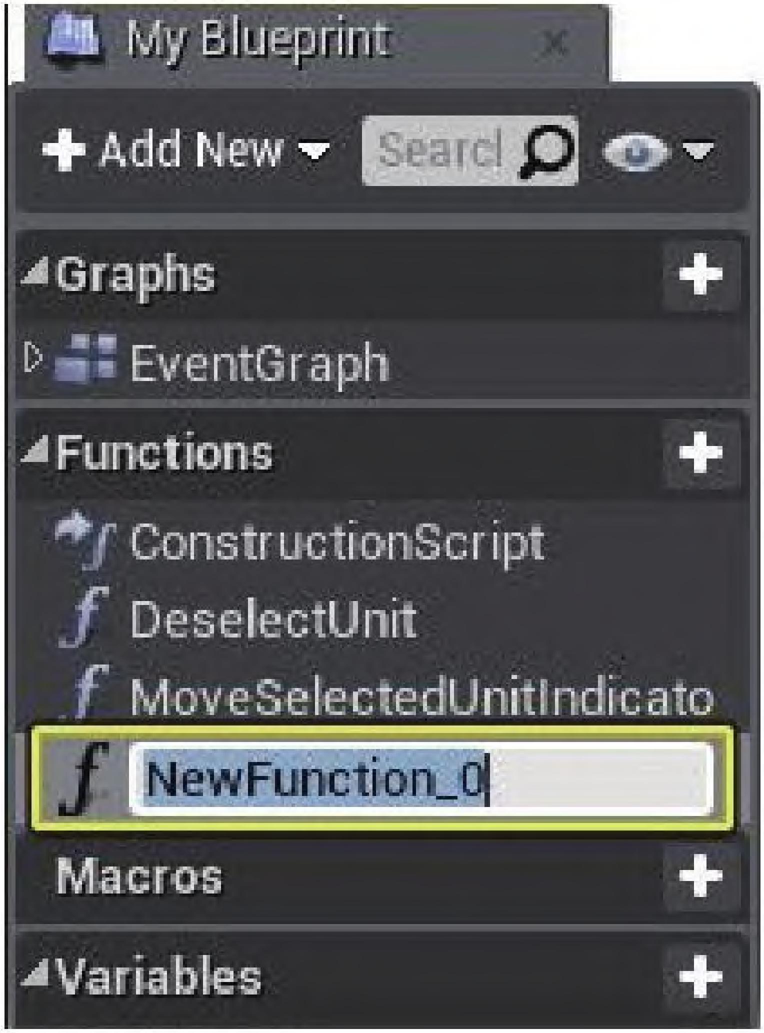



图9-52

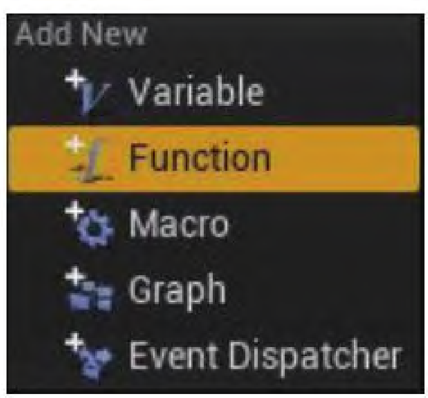



图9-53

### 2. 在蓝图接口中创建函数

在Blueprint Interface（蓝图接口）中创建函数同在Class Blueprint（类蓝图）或Level Blueprint（关卡蓝图）中创建函数一样，但是实现是有很大差别的。

要想在蓝图接口中创建一个函数：

**Step 1** 在Graph（图表）模式，从My Blueprint（我的蓝图）选卡下，单击New Function（新建函数）按钮，如图9-54所示。

**Step 2** 输入你的函数的名称，如图9-55所示。

将在蓝图编辑器的Graph Editor（图标编辑器）选卡的一个新选卡中打开你的函数。你也可以在我的蓝图选卡中右击并选择Function（函数）来创建一个函数，如图9-56所示。

图9-54

图9-55

图9-56

注意：Blueprint Interfaces（蓝图接口）中的函数没有图表，因为它们没有实现。在蓝图接口中创建一个新函数时打开的图表将是空的，只有函数的输入节点，且该节点呈现为灰掉状态。

## 9.3.4 编辑函数

一旦创建了函数，你就需要定义其功能。这个过程需要两步：

•创建必要的输入和输出参数。

•在你的输入和输出之间创建节点网络来定义函数行为。

你可以在详细信息选卡中设置Description（描述）、Category（类别）、Access Specifier（访问修饰符）及该函数是否为Pure（纯函数）。

打开你的函数的详细信息面板，你可以：

•在My Blueprint（我的蓝图）选卡中选择你的函数，如图9-57所示。

图9-57

• 在调用函数的图表中选中该函数，如图9-58所示。

图9-58

•在函数的图表中选择函数入口（或结果）节点，如图9-59所示。

图9-59

### 1. 输入和输出参数

你也可以在你的函数的详细信息面板中设置输入和输出参数。

要想添加输入或输出参数：

**Step 1**　单击详细信息面板的Inputs（输入）或Outputs（输出）部分的New（新建）按钮。

**Step 2**　命名该新参数并使用下拉菜单设置其类型。注意，和蓝图宏不同，你仅可以向函数添加数据输入和输出。在这个示例中，有两个向量型数据输入VectorA和VectorB，还有一个浮点型数据输出Distance，如图9-60所示。

函数图表中的入口节点和结果节点将自动地更新正确的引脚，同时任何函数调用节点也会进行适当的更新，如图9-61和图9-62所示。

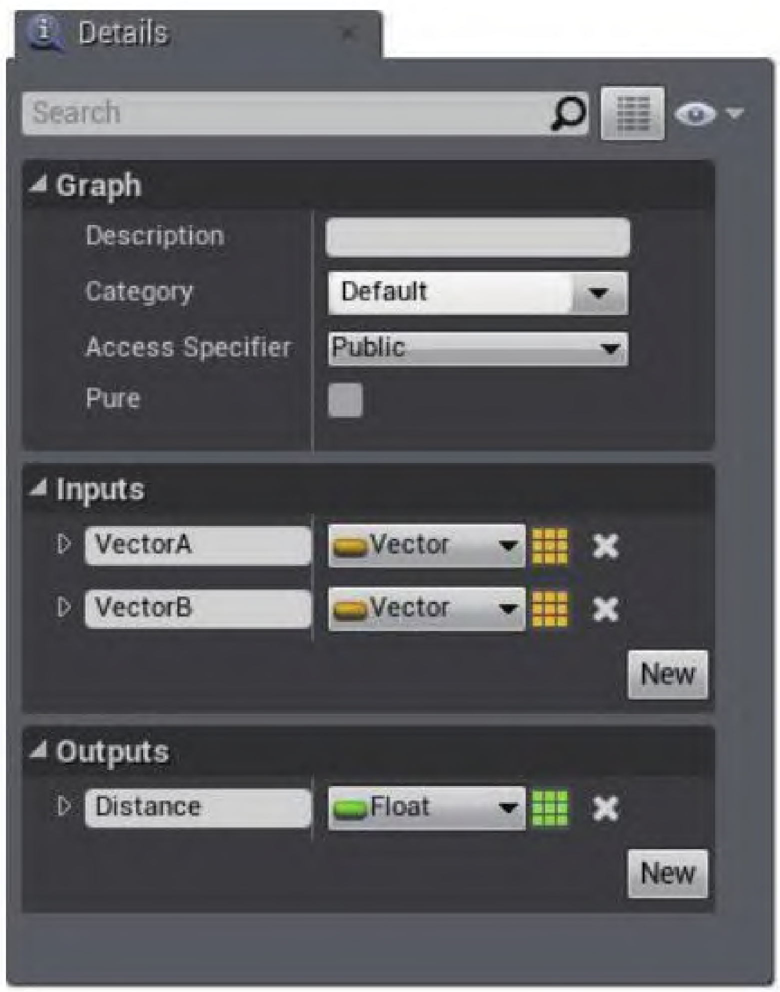



图9-60

图9-61

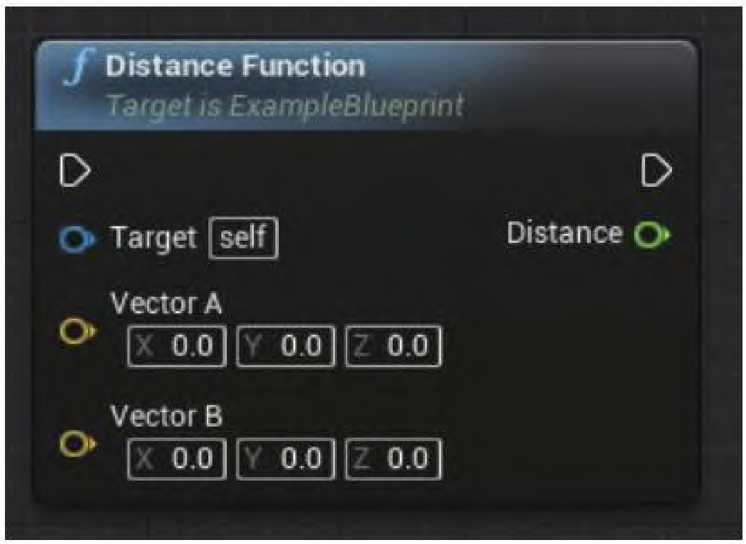



图9-62

Step 3　你也可以通过展开参数的项来设置默认值，并选择是否应该通过引用传递该值，如图9-63所示。

要想改变节点边上的这个参数的引脚的位置，请使用展开的详细信息面板项的向下和向上箭头。

现在，可以定义你的函数的功能了。我们将通过在入口节点和结果节点之间创建节点网络来完成这个处理。

2. 定义功能

你可以通过创建连接输入节点和输出节点的蓝图图表来定义你的函数。在这个示例中，我们将创建必要的节点网络来应用勾股定理的三维版本，返回三维空间中两个点间的距离。

```
dx=(x2-x1)^2
dy=(y2-y1)^2
dz=(z2-z1)^2

D=sqrt(dx+dy+dz)
```

如图9-64所示。

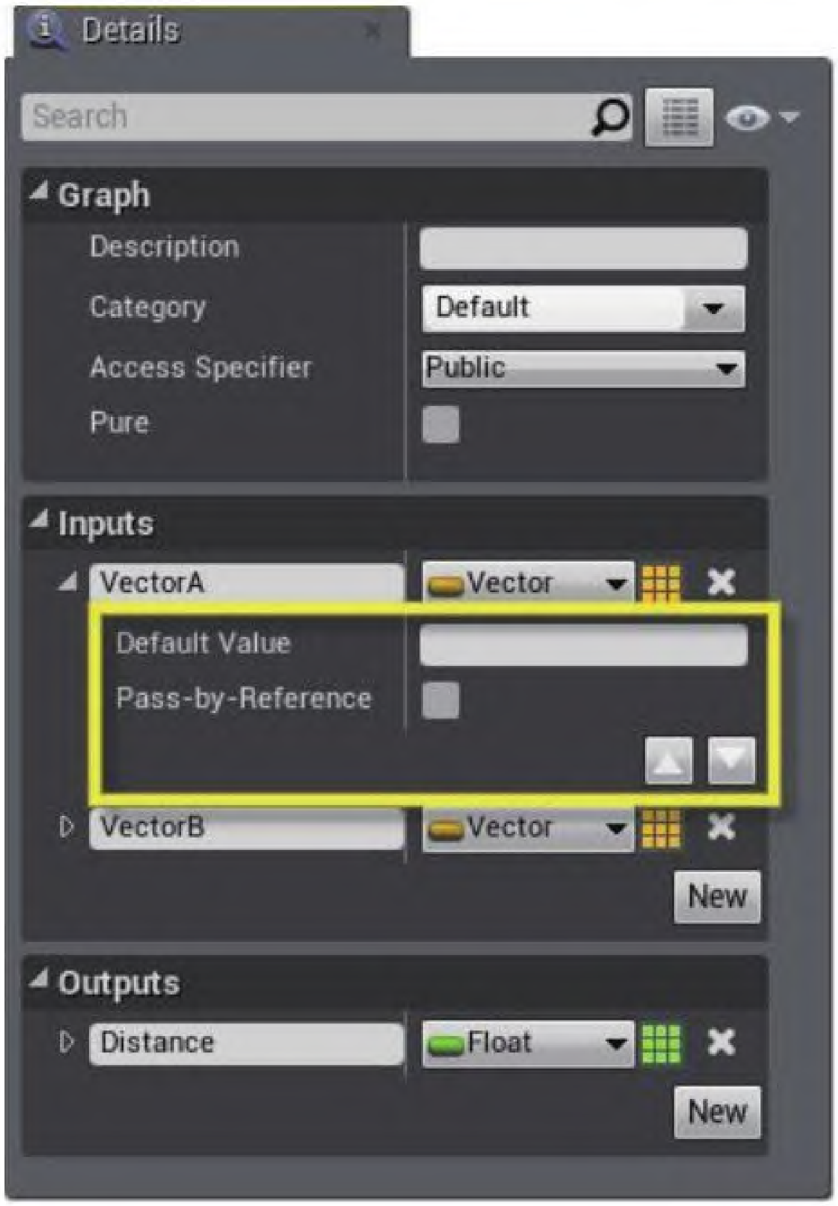

Details
Search
Graph
Description
Category
Default
Access Specifier
Public
Pure
Inputs
VectorA
Vector
Default Value
Pass-by-Reference
VectorB
Vector
New
Outputs
Distance
Float
New

图9-63

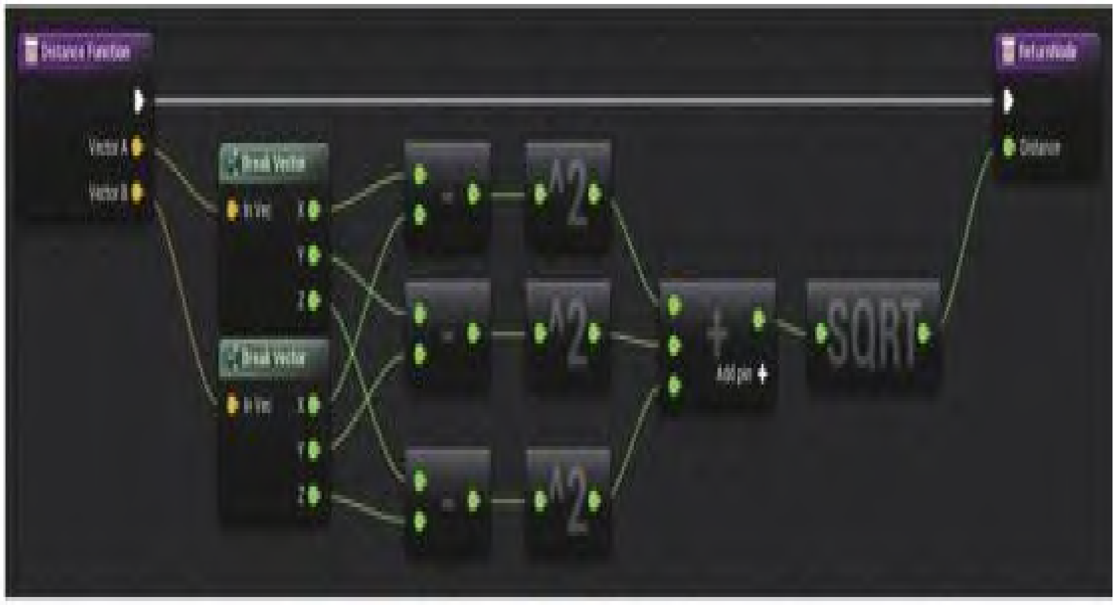



图9-64

## 9.3.5　调用函数

一旦你创建并定义了你的函数后，就可以在你的事件图表中调用它。要想创建一个调用你的函数的节点：

•从我的蓝图选卡中拖动你的函数到事件图表内的空白处。

•在事件图表中右击，或者从适当的执行引脚或数据引脚拖动鼠标来打开关联菜单。在关联菜单中搜索你的函数，然后选中它来添加函数调用节点。

这是个简单的网络，取入了两个向量变量，每次更新时，将计算这两个变量间的距离，并向屏幕输出该距离值，如图9-65所示。

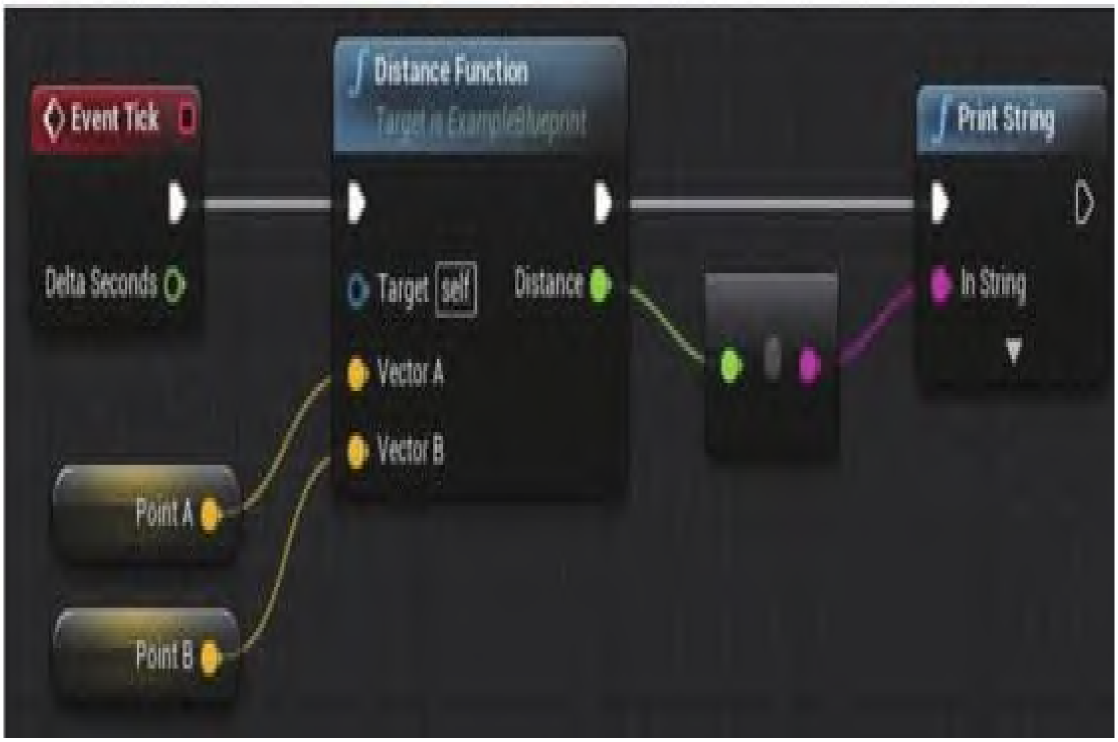



图9-65

在这个示例中，我们创建了两个公有向量型变量。在每个向量的设置中，我们设置Show3D Widget（显示3D控件）为True。现在，在将该蓝图添加到关卡中后，3D控件将会出现在由点A和点B定义的位置处。该3D控件允许我们通过在视口中移动点A和点B，来轻松地改变它们的值，如图9-66所示。

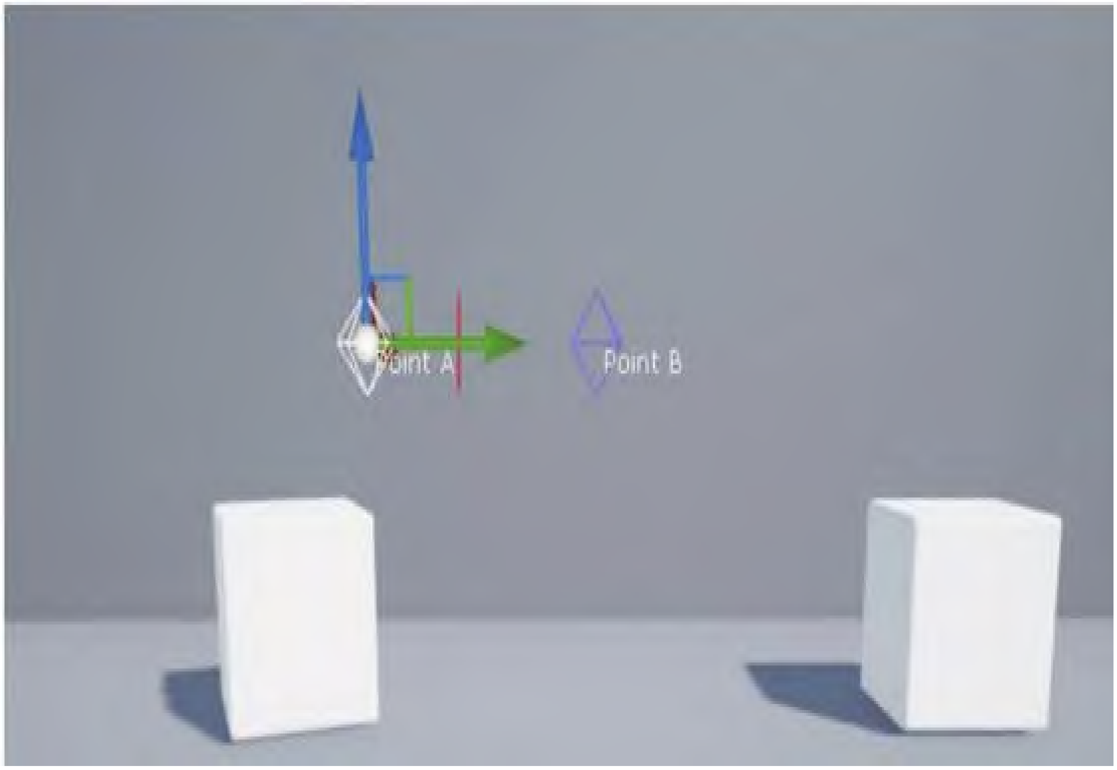



图9-66

现在，当我们测试地图时，将在每次更新时记录两个点之间的值，以证明我们的函数是有效的，如图9-67所示。

图9-67

## 9.3.6　函数疑难解答

如果在函数调用节点上看到警告“！”条，提示“不能找到名称为（FunctionName）的函数”信息时，请编译你的蓝图，如图9-68所示。

图9-68

如果你改变你的函数上的输入或输出参数的数量，你也需要编译你的蓝图。

## 9.3.7　函数调用

Function Calls（函数调用）是可以在蓝图中形成的操作，该蓝图与属于目标Actor或对象的函数相对应。对于关卡蓝图来说，相关Actor很多时候都是关卡蓝图本身。函数调用显示为一个盒形，具有显示了函数名称的标题。不同类型的函数调用具有不同颜色的标题。

Other Function Calls（其他函数调用）是属于除了该蓝图之外的其他对象或Actor的函数。例如，该蓝图可能有一个StaticMeshComponent，该StaticMeshComponent可以通过SetStaticMesh函数调用修改它的网格物体。由于这个函数属于StaticMeshComponent而不属于蓝图，所以它是Other Function Call（其他函数调用）。

Pure Function Calls（纯函数调用）是可以执行的特殊动作，它不会直接影响世界或者世界中的对象。这些函数一般执行类似于这样的事情：输出一个属性值、数学运算操作（比如两个值间的加减乘除等），所产生的结果

不会对任何事物造成影响。只有对该结果进行赋值或使用该结果，才能对世界产生影响。

| | |
|---|---|
| | |
| 标准 | 压缩 |

9.4

# 宏

蓝图宏或宏从本质上讲和合并的节点图表一样。它们具有由通道节点指定的一个入口点和一个出口点。每个通道可以具有多个执行引脚或数据引脚，当在其他蓝图及图表中使用该宏节点时这些引脚会呈现在宏节点上。

## 9.4.1 创建蓝图宏

蓝图宏可以在类蓝图或关卡蓝图中进行创建，就像蓝图函数一样。蓝图宏还可以放到蓝图宏库中。

**1.在另一个蓝图中创建蓝图宏**

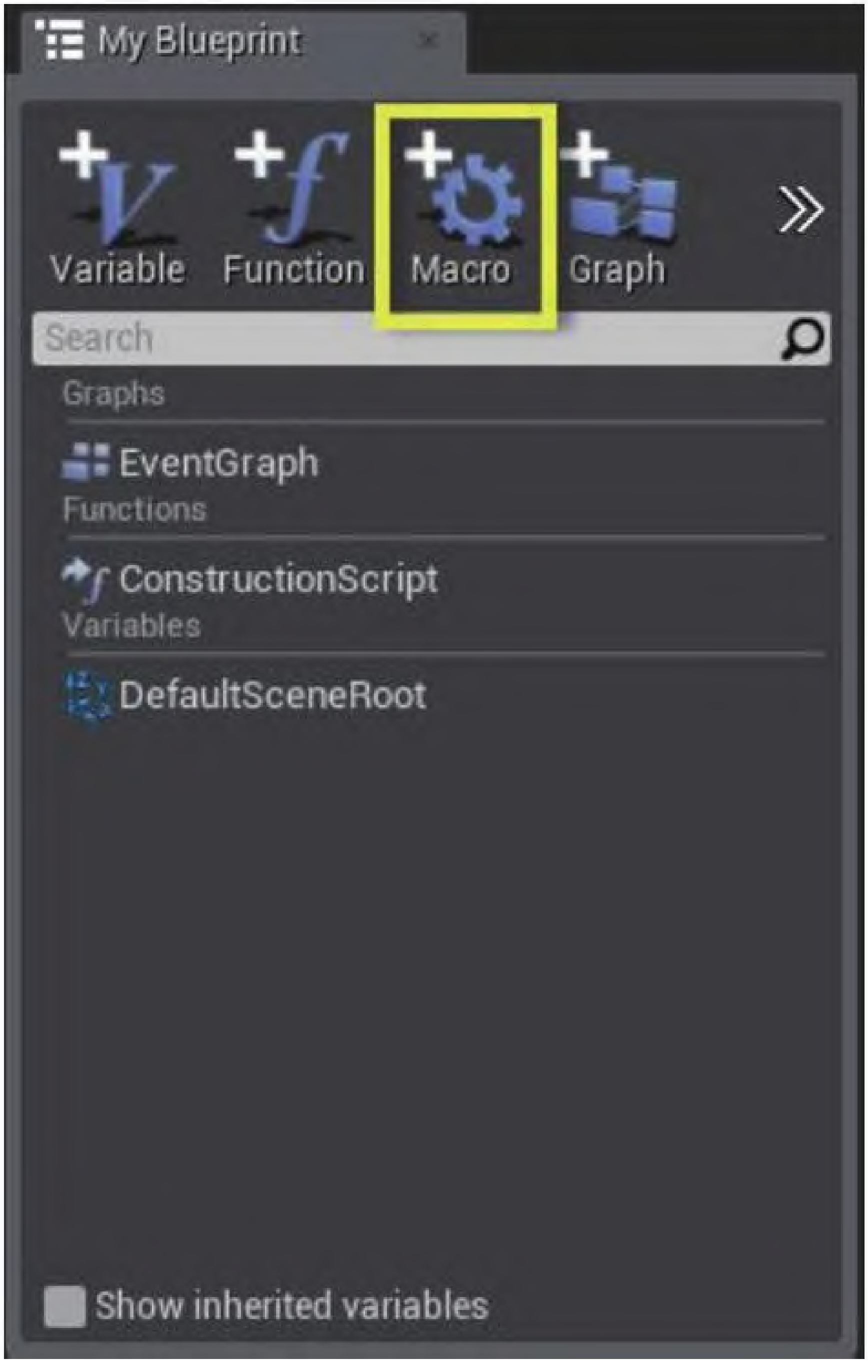
要想在一个类蓝图或关卡蓝图中创建蓝图宏：

**Step 1** 在Graph（图表）模式，从My Blueprint（我的蓝图）选卡下，单击New Macro（新建宏）按钮，如图9-69所示。

**Step 2** 输入你的Blueprint Macro（蓝图宏）的名称，如图9-70所示。

将在蓝图编辑器的Graph Editor（图标编辑器）选卡的一个新选卡中打开你的蓝图宏。

你也可以在我的蓝图选卡中右击并选择Macro（宏）来创建一个蓝图宏，如图9-71所示。


My Blueprint
Variable
Function
Macro
Graph
Search
Graphs
EventGraph
Functions
ConstructionScript
Variables
DefaultSceneRoot
Show inherited variables
图9-69

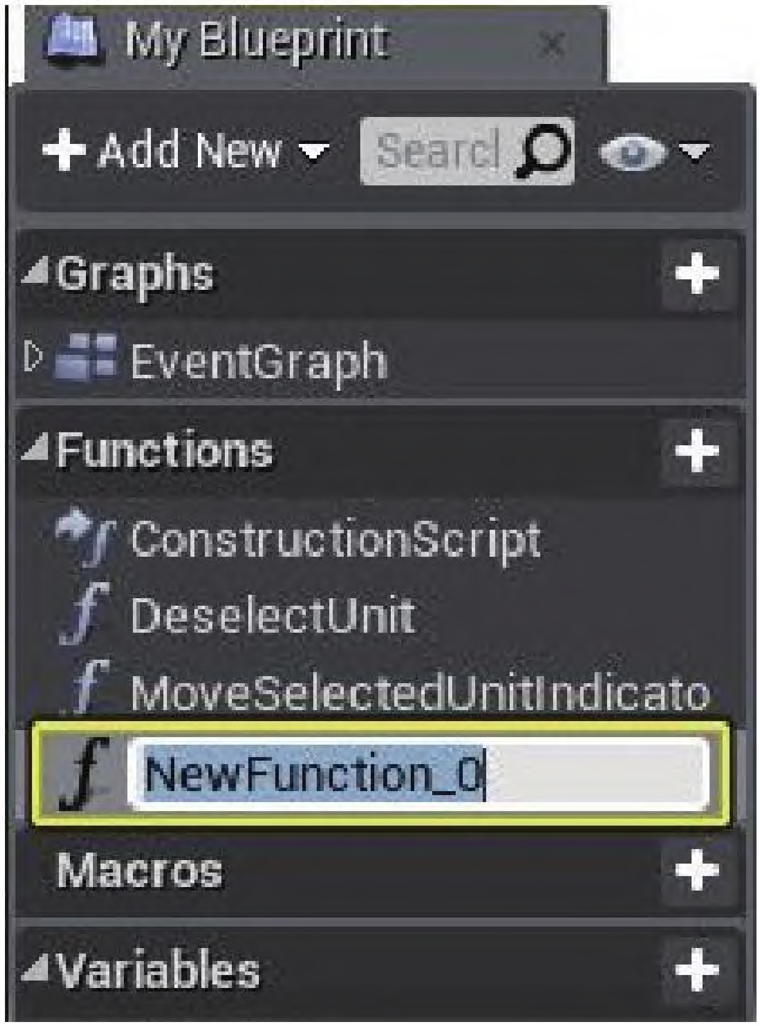

My Blueprint
Add New
Searcl
Graphs
EventGraph
Functions
ConstructionScript
DeselectUnit
MoveSelectedUnitIndicato
NewFunction_0
Macros
Variables

图9-70

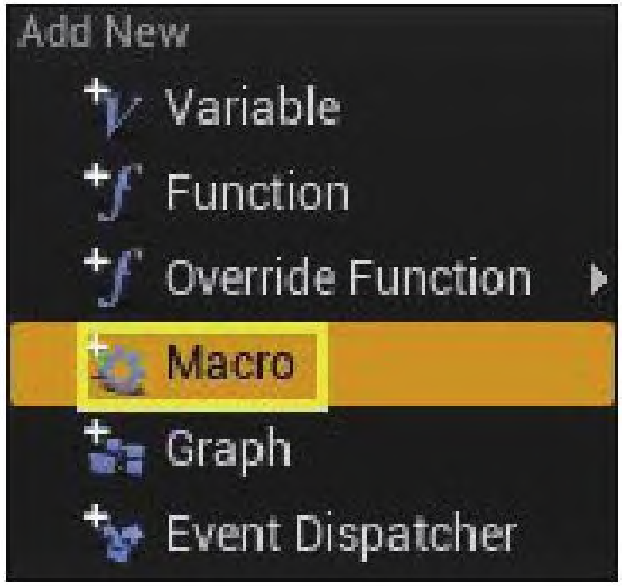



图9-71

2. 在宏库中创建蓝图宏

在蓝图宏库中创建的蓝图宏可以用于你的项目中的任何蓝图内。

要想在蓝图宏库中创建一个蓝图宏：

**Step 1** 在Graph（图表）模式，从My Blueprint（我的蓝图）选卡下，单击New Macro（新建宏）按钮，如图9-72所示。

**Step 2** 输入你的Blueprint Macro（蓝图宏）的名称，如图9-73所示。

将会在蓝图编辑器的Graph（图表）面板的一个新选卡中打开你的蓝图宏。

你也可以在我的蓝图面板中右击并选择Macro（宏）来创建一个蓝图宏，如图9-74所示。

图9-72

My Blueprint
Macro
Search
Macros
NewMacro
Show inherited variables

图9-73

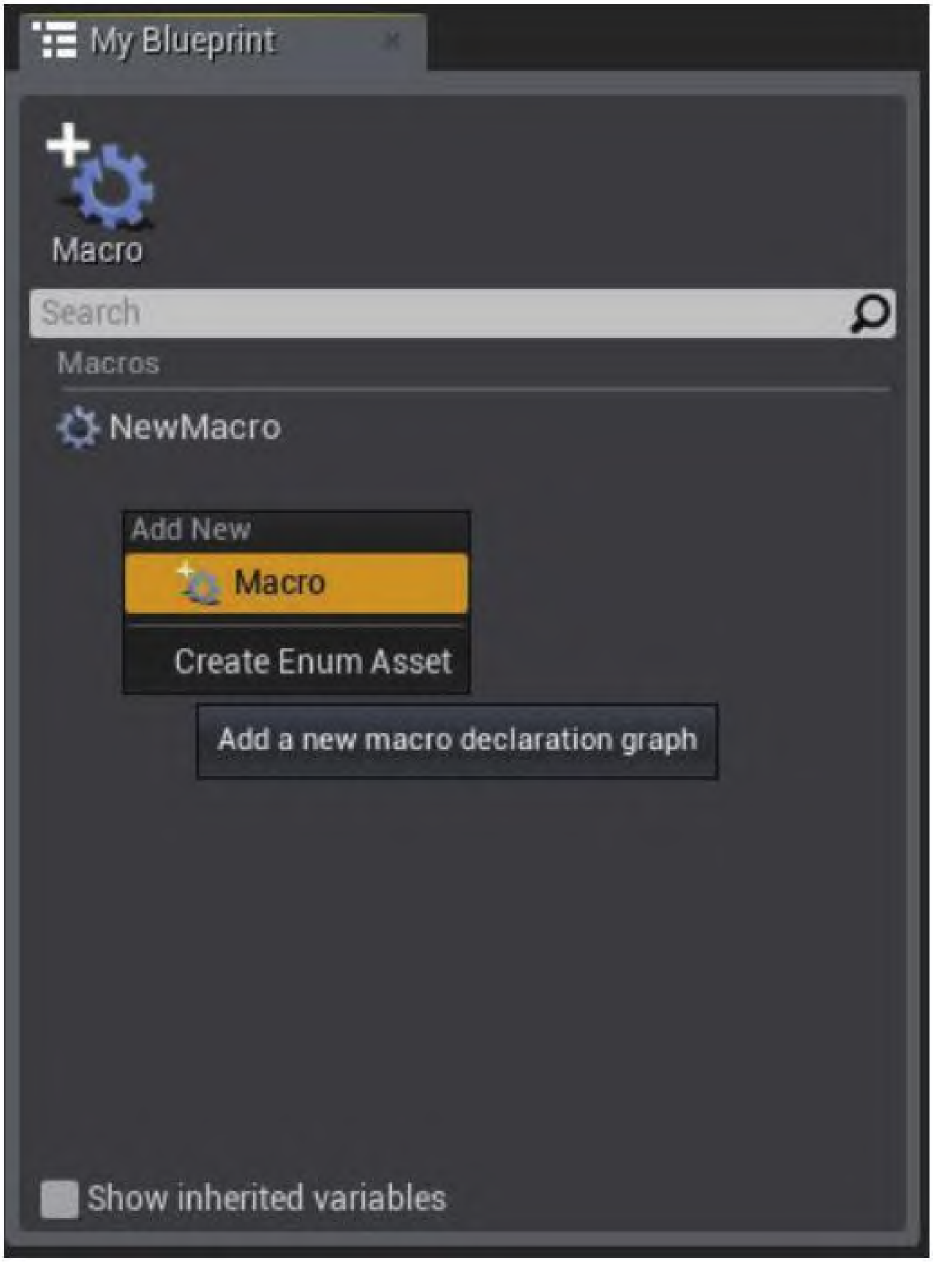



图9-74

## 9.4.2　构建蓝图宏

当你第一次创建一个蓝图宏时，将会打开一个新图表，该图表包含一个Inputs（输入）通道节点和一个Outputs（输出）通道节点，如图9-75所示。

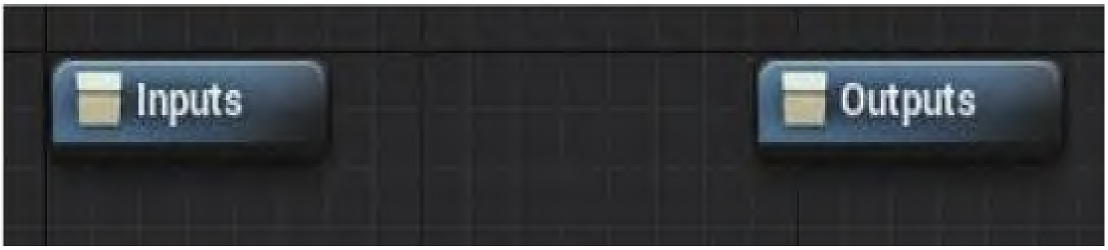



图9-75

在你的蓝图宏的Details（详细信息）面板中，你可以添加输入和输出执行引脚及数据引脚。你也可以设置你的蓝图宏的Description（描述）、Category（类别）及Instance Color（实例颜色），如图9-76所示。

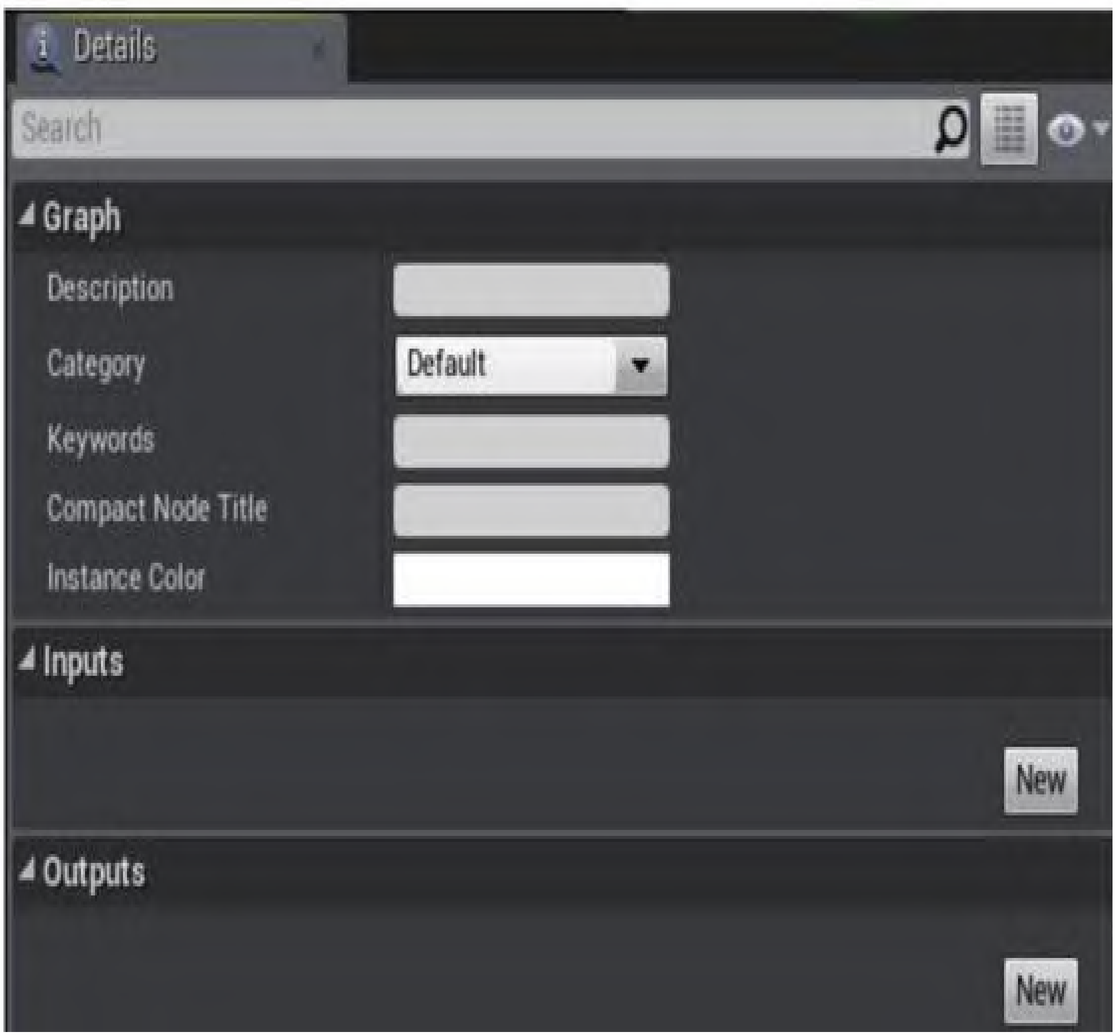



图9-76

要想添加输入或输出参数：

**Step 1**　单击详细信息面板的Inputs（输入）或Outputs（输出）部分的New（新建）按钮。

**Step 2**　命名该新参数并使用下拉菜单设置其类型。在这个示例中，有一个整型数据输入参数Score、一个输入执行引脚Test及两个输出执行引脚Win和Lose，如图9-77所示。

蓝图宏图表中的通道节点将自动地更新为正确的引脚，如图9-78所示。

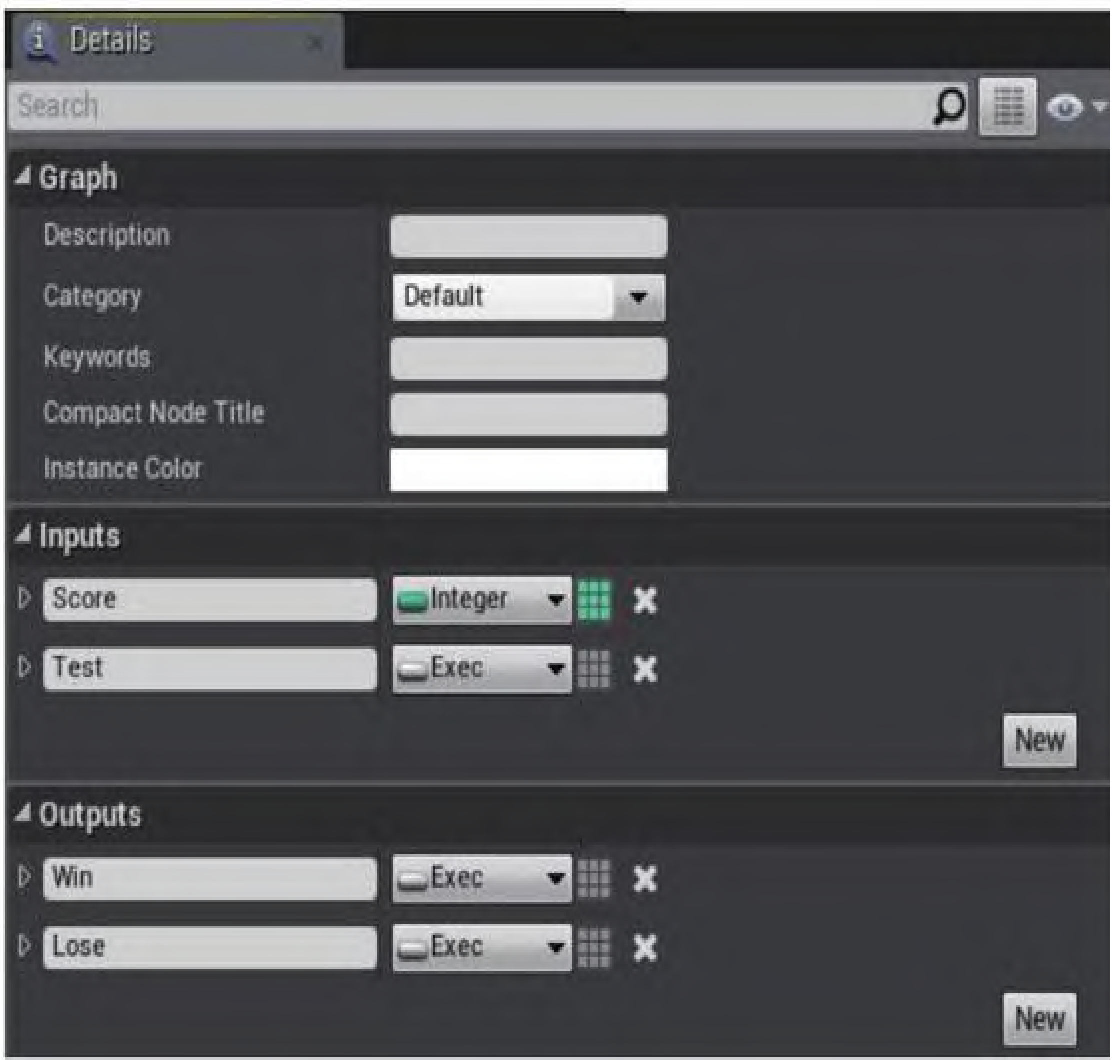



图9-77

图9-78

**Step 3** 你也可以通过展开参数项来设置一个默认值，如图9-79所示。

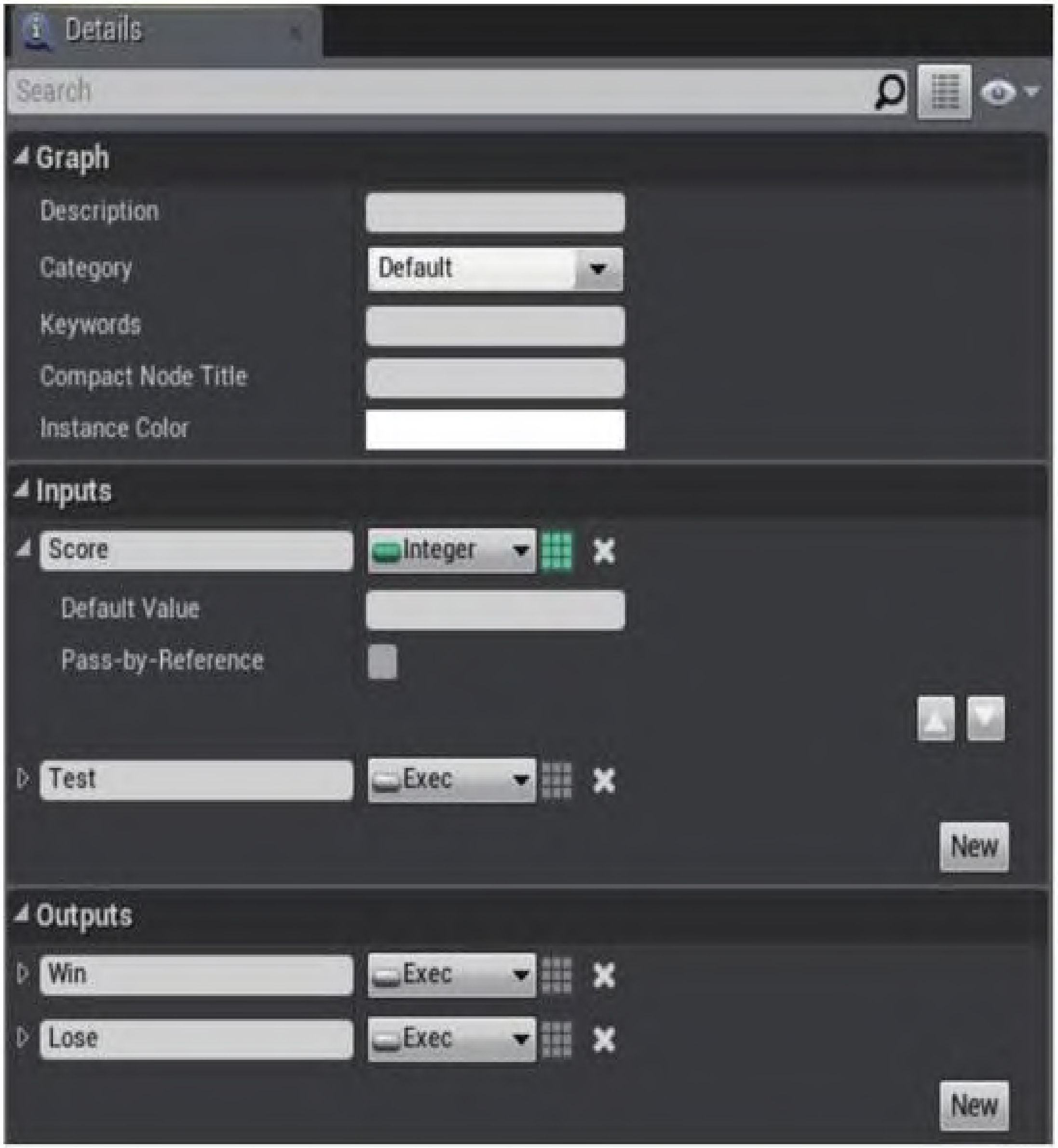



图9-79

要想改变节点边缘上的这个参数的引脚的位置，请使用展开的详细信息面板项的向下和向上箭头。

要想给你的蓝图宏设置一些功能，请把数据引脚及执行引脚连接到你的Inputs和Outputs通道节点，并在它们之间创建一个节点网络，如图9-80所示。

这个示例蓝图宏判断输入宏中的分数是否大于获胜的必要分数，然后根据比较结果触发不同的输出执行流程。注意，这里通过使用详细信息面板中

的向上和向下箭头翻转了Test和Score引脚，以避免在蓝图宏图表中出现交叉连线，如图9-81所示。

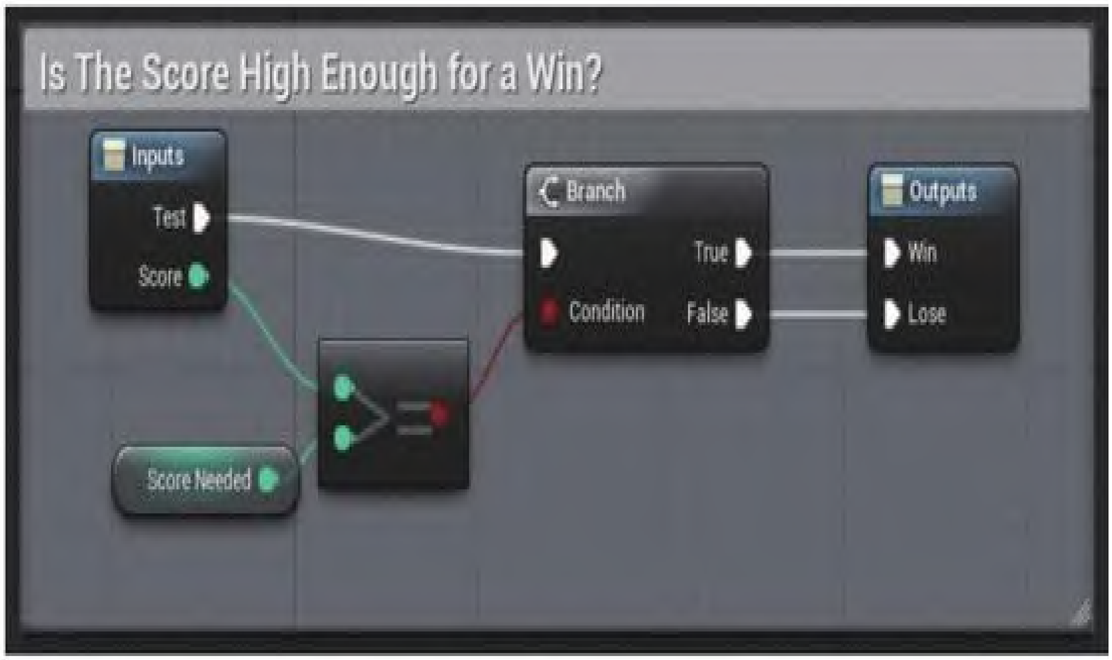



图9-80

图9-81

和函数不同，宏可以具有多个输出执行引脚，所以你可以具有类似这样的执行流程：根据图表逻辑的结果执行不同的输出执行引脚。同时，只要宏中的节点不是执行节点，你也可以具有没有执行引脚的宏，这样的宏仅简单地操作数据，如图9-82所示。

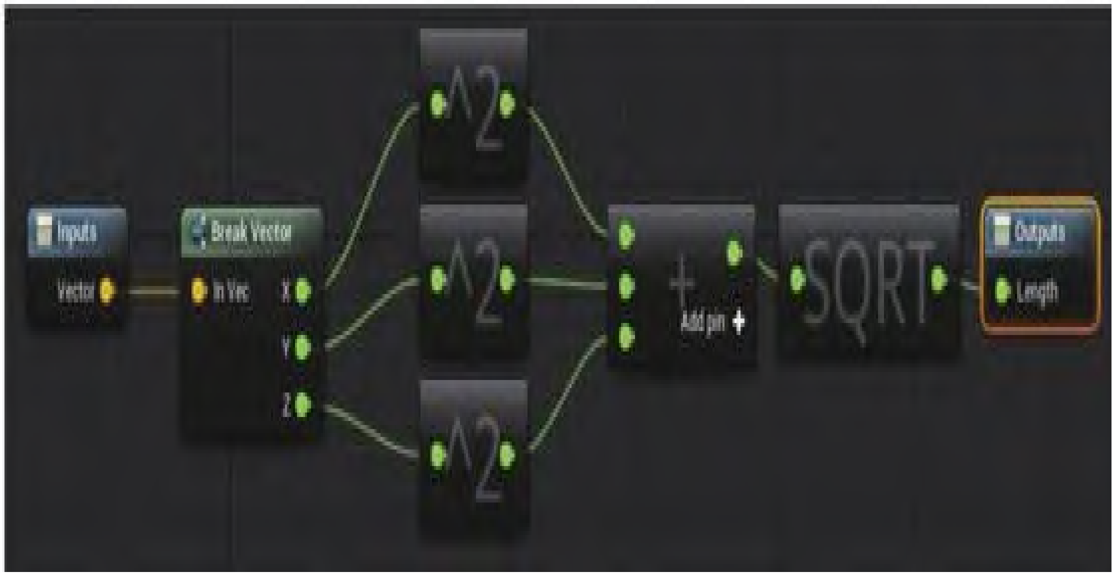



图9-82

实现宏：

有很多种把宏节点添加到另一个蓝图图表中的方法。和Function（函数）节点及Custom Event（自定义事件）调用节点一样，你也可以在你的蓝图中添加多个宏节点的副本。

要想添加一个宏，只需右击图表并在弹出的关联菜单中选择Macro（宏）即可，如图9-83所示。

图9-83

你可以从另一个节点的引脚处拖动并释放鼠标，如果该节点具有相应的变量类型和流程方向，那么该宏将出现在关联菜单中，如图9-84所示。

图9-84

当然，你可以从我的蓝图选卡中拖动该宏并把它放置到图表中，如图9-85和图9-86所示。

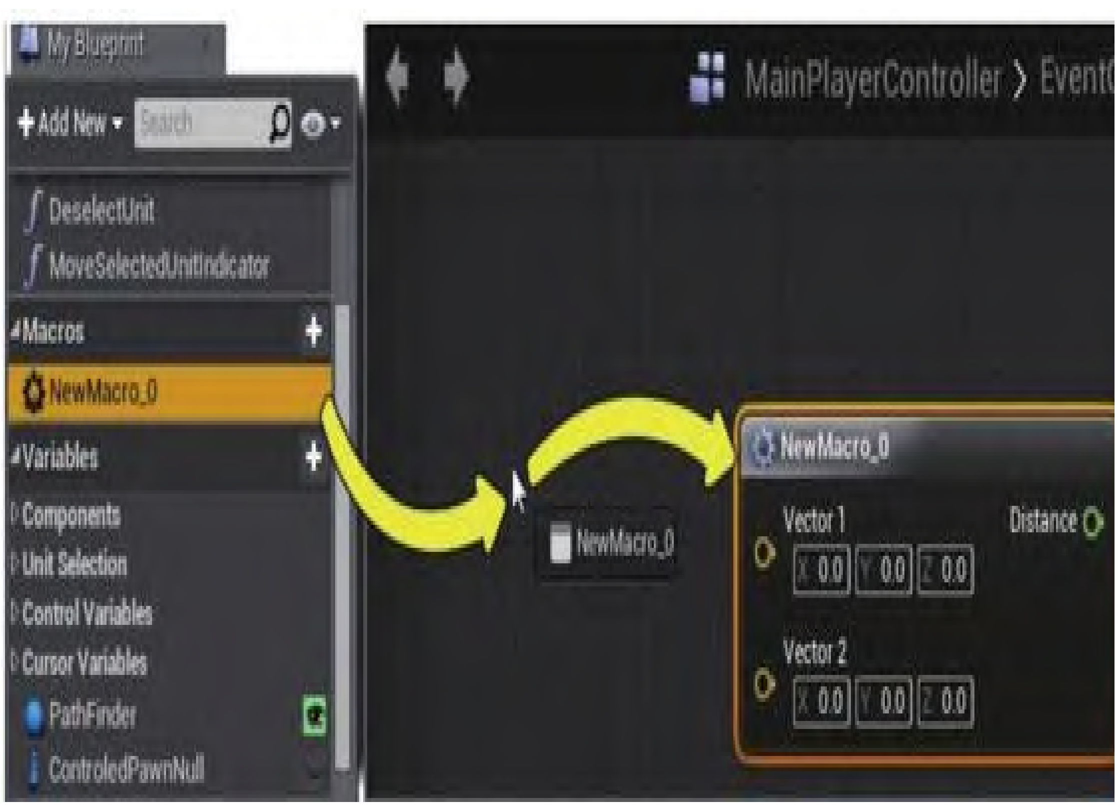



图9-85

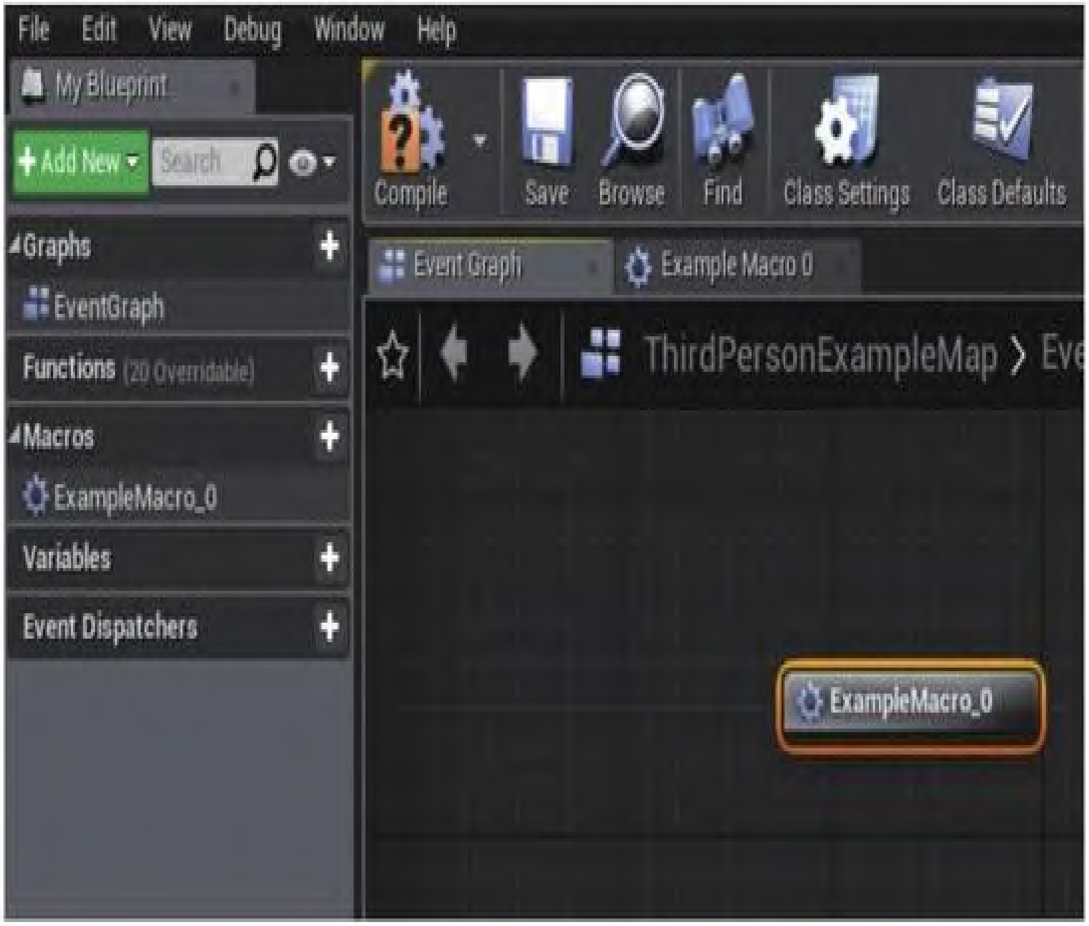



图9-86

一旦将宏节点添加到了图表中，它的处理就和其他任何节点一样，输入和输出引脚可以相应地进行连接。双击宏节点将会打开宏的图表。

9.5

# 数学表达式节点

## 9.5.1　概述

要想创建一个数学表达式节点，请右击图表并从关联菜单中选择Add Math Expression（添加数学表达式），如图9-87所示。

数学表达式节点就像一个合并的图表。它是一个独立的节点，你可以双击它来打开构成其功能的子图表。最初，该名称/表达式是空的。任何时候，当你重命名该节点时，都将会解析新表达式并生成新的子图表。

要想重命名该节点，并更新子图表，请右击该节点，并选择Rename（重命名）。

比如，你可以使用以下表达式来重命名该数学表达式节点：

（1+x）*sin（myVar）-2.4/rand（）

这将会更新该数学表达式节点，使其具有两个浮点型输入X和MyVar，及一个浮点型输出，如图9-88所示。

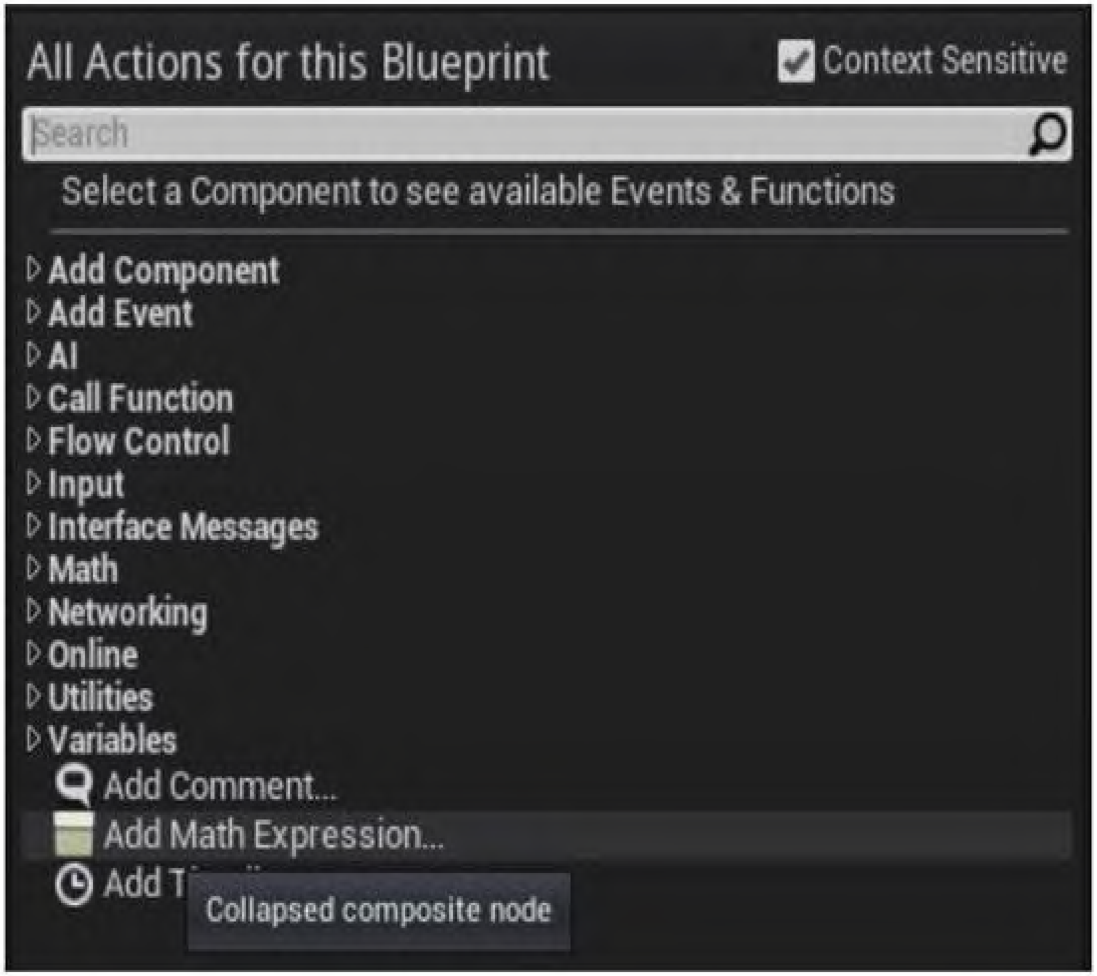



图9-87

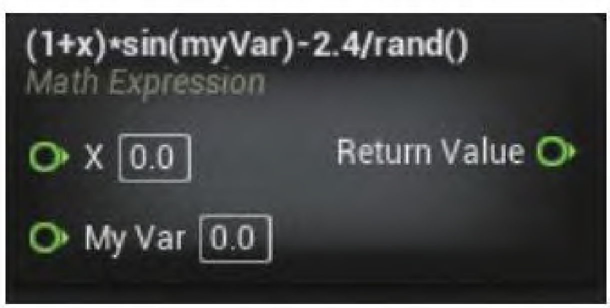



图9-88

如果你双击具有该表达式的节点，那么你将看到以下这个子图表，如图9-89所示。

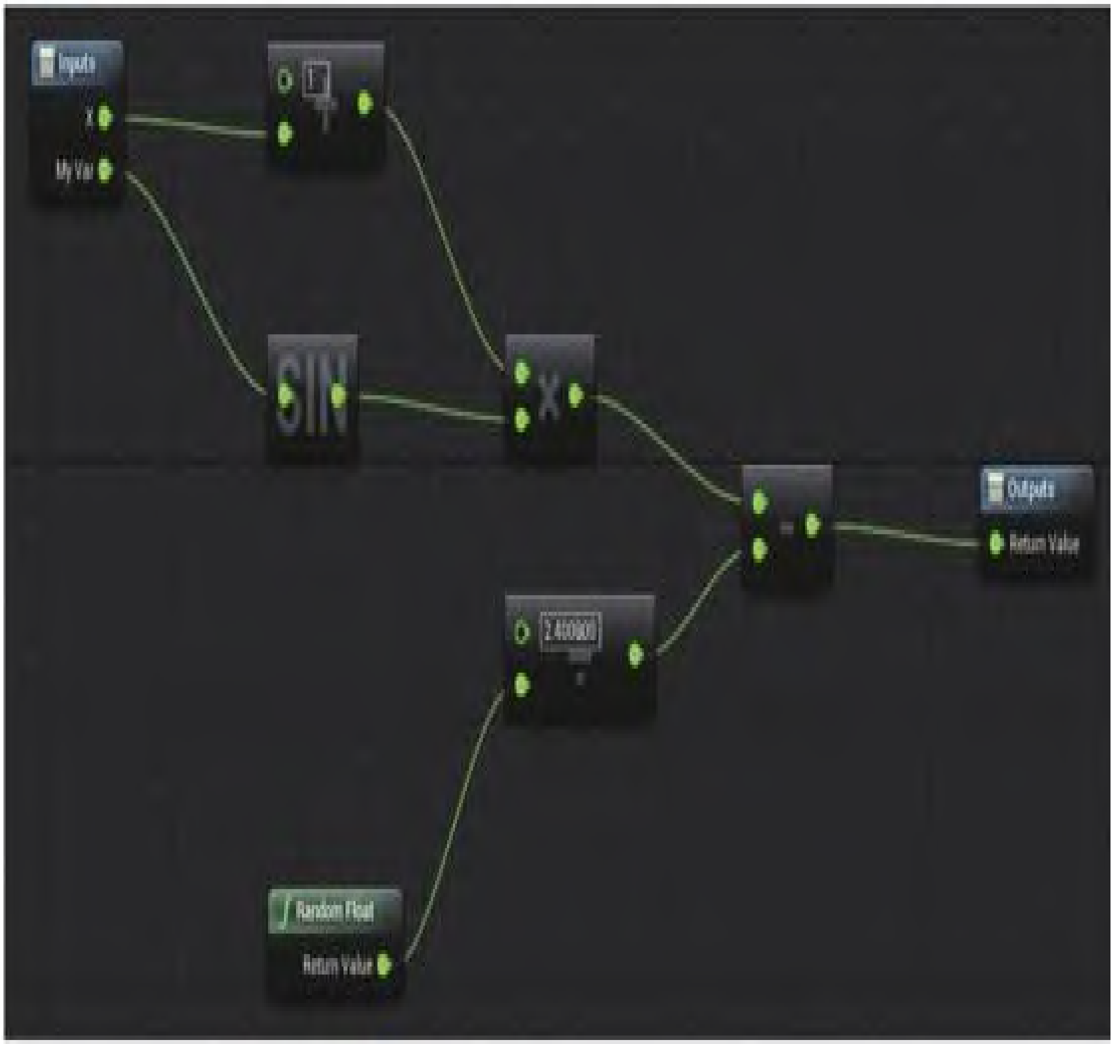



图9-89

子图表是根据以下规则创建的：

**Step 1**　字母名称（以字母开头的符号）应该变为数学表达式上的变量、输入引脚，或者函数调用。上面的示例中字母名称有：x、sin、myVar和rand。

**Step 2**　如果一个字母名称和BSlueprint中的一个现有变量一样，那么它将变成一个变量获取节点（也就是如果蓝图已经具有了myVar变量）。

**Step 3**　如果字母名称后面有一个左括号（比如sin（和rand（），那么它将会变成一个函数节点。

**Step 4**　如果之前的两种情形都不符合，那么该字母名称将变成数学函数节点上的一个浮点型输入。

**Step 5**　数值常量将总是变成引脚输入。它们自身永远不会创建一个节点，但是会使用它们来填充其他节点的输入文本域。

**Step 6**　数学运算符（像+、×、-）会变成函数节点。

**Step 7**　和数学运算中的处理一样，括号内的表达式具有优先权，需要优先计算。

### 9.5.2　变量

变量命名非常灵活，但是记住以下几点非常重要：

•变量名称本身可以包含数字，但是不能以数字开头。

•变量名称不能和隐藏的蓝图变量名称一样。

•确保你正在使用正确的变量类型。比如，boolVar+1.5就不是一个有效的表达式。

### 9.5.3　运算顺序

运算顺序如下所示（按照优先级从高到底的顺序）：

（1）括号内的表达式。

（2）阶乘。

（3）指数。

（4）乘除法。

（5）加减法。

### 9.5.4　可用的蓝图函数

代码函数库中的蓝图纯函数应该都是可用的。这包括：

•三角函数（sin/cos/tan、asin、acos，等）：

sin（x）。

•区间限定函数（min，max，clamp，等）：

clamp（y，0，20）。

•四舍五入函数（floor，fceil，round等）：

round（z）。

•指数函数（square，power，sqrt，loge，e/exp等）。

某些函数具有多个经常使用的名称。为了处理这个问题，同一个函数或节点会有几个相关联的别名。一些示例如下：

•Power（幂数）别名：power，pow。

•三角函数（asin/arcsin，acos/arccos，等）。

因为你正在输入函数，而不是连接引脚，所以请确保输入正确的数值及参数类型。如果数值和参数类型不匹配，那么该数学该表达式节点将显示一个错误。

### 9.5.5 基本结构体类型

在数学中，有一些经常要用到的基本结构体，如向量、变换等。你可以在表达式中轻松地创建及操作这些类型。

•vector关键字生成一个MakeVector节点：

vector（x，y，z）。

•rotator关键字生成一个MakeRotator节点：

rotator（x，y，z）。

•transform关键字生成一个MakeTransform节点：

transform（vec（x，y，z），rot（p，y，r），vec（6，5，4））。

正如某些函数有别名一样，向量和旋转量也有别名。

•Vector（向量）别名

vector，vec，vect，等。

•Rotator（旋转量）别名

rotator，rot等。

### 9.5.6　支持的操作符

支持以下所有操作符，这些操作符可以同逻辑操作符和比较操作符结合使用，来创建复杂的表达式。

•乘除法操作符：

*，/，%。

•加减法操作符：

+，-。

•关系操作符：

<，>，<=，>=。

•等于：

==，！=。

•逻辑操作符：

||，&&，^。

### 9.5.7　不支持的操作符

目前，不支持以下操作符。

•条件操作符：

?：。

•位操作符：

|，&，<<，>>。

•一元前缀操作符：

+，-，++，--，～，！等。

- 后缀操作符：

++，--，[]等。

9.6

# 节点

## 9.6.1 概述

Nodes（节点）是指可以在图表中应用其来定义特定图表及其包含该图表的蓝图的功能的对象，比如事件、函数调用、流程控制操作、变量等。

## 9.6.2 应用节点

尽管每种类型的节点执行一种特定的功能，但是所有节点的创建及应用方式都是相同的。这有助于在创建节点图表时提供一种直观的体验。

## 9.6.3 放置节点

通过从关联菜单中选择一种节点类型，可以把新节点添加到图表中。关联菜单中所列出的节点类型，根据访问该类型列表的方式及当前选中的对象的不同而有所差别。

•在图表编辑器选卡中，右击空白区域，会弹出可以添加到图表中的所有节点的列表。如果选中一个Actor，那么将会列出那种类型的Actor支持的事件，如图9-90所示。

•从节点的一个引脚处拖动鼠标产生一个连接并在空白处释放鼠标，会弹出一个节点列表，这些节点具有和连接的起始引脚相兼容的引脚，如图9-91和图9-92所示。

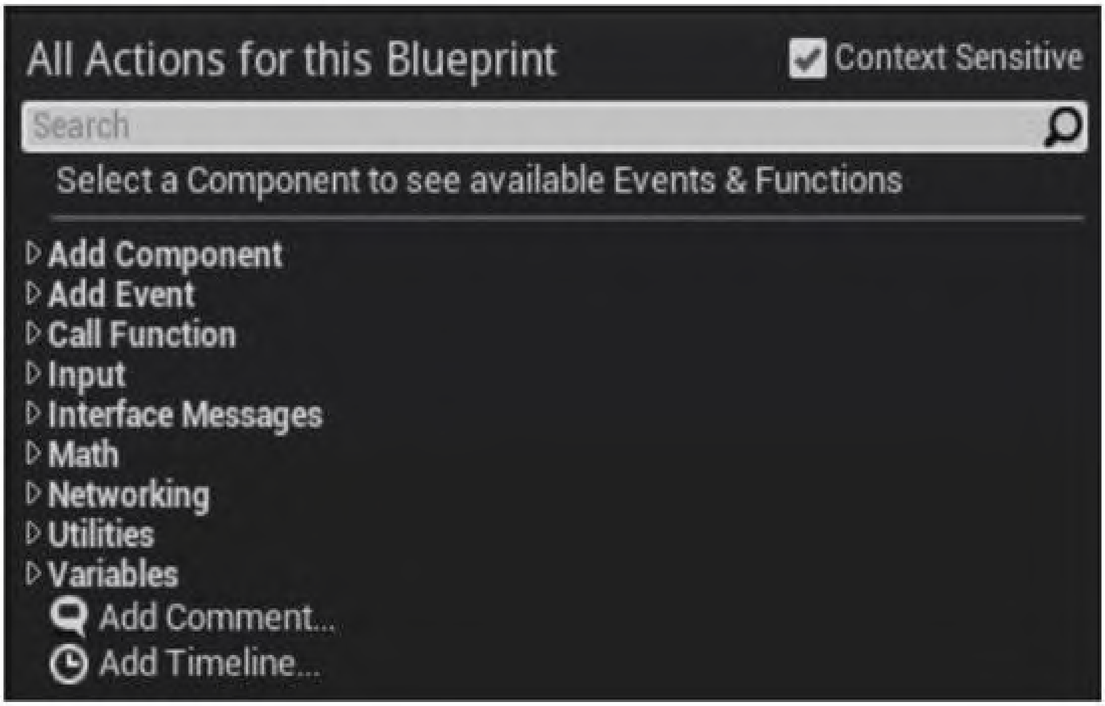

All Actions for this Blueprint
Context Sensitive
Search
Select a Component to see available Events & Functions
Add Component
Add Event
Call Function
Input
Interface Messages
Math
Networking
Utilities
Variables
Add Comment...
Add Timeline...

图9-90

OnTakePointDamage (unreal_holo_6)
Place a new node.
Damage
Instigated By
Hit Location
FHit Component
Bone Name
Shot from Direction
Damage Type
Damage Causer

图9-91

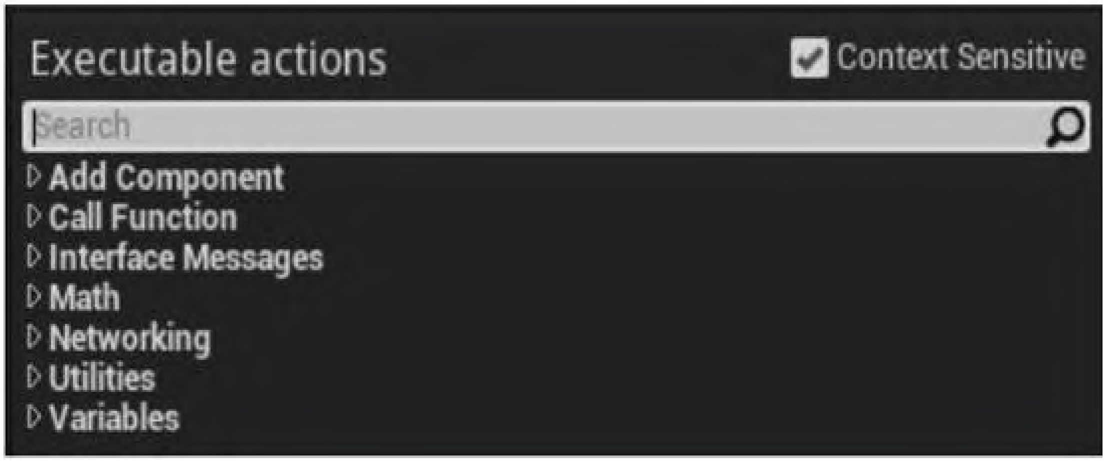



图9-92

## 9.6.4　选择节点

在图表编辑器选卡中单击一个节点，可以选中该节点，如图9-93所示。

图9-93

通过按住Ctrl键并单击节点，可以将节点添加到当前的选中项或者将其从当前的选中项删除，如图9-94所示。

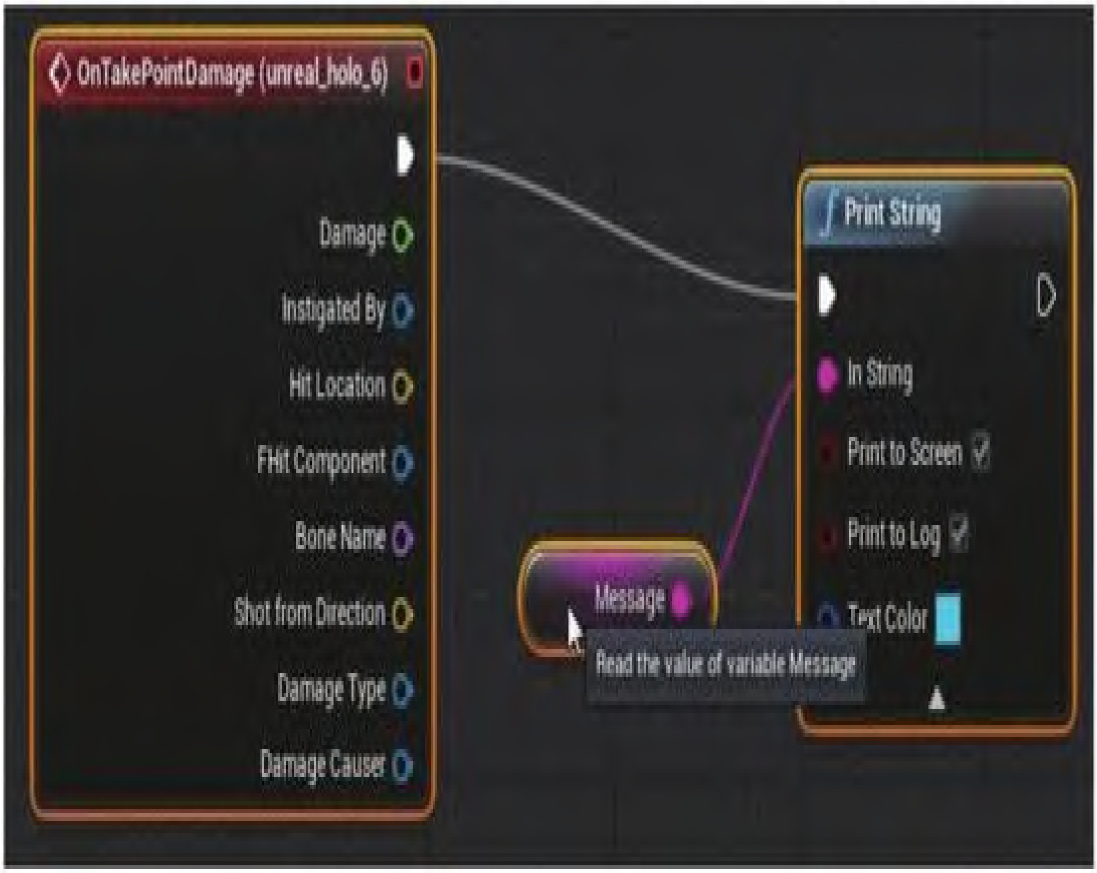



图9-94

通过单击并拖动鼠标创建一个区域选择框，可以选中多个节点。按住Ctrl键+单击并拖动鼠标创建一个区域选择框，可以切换对象的选中状态。按住Shift键+单击并拖动鼠标创建一个区域选择框，可以把选择框中的对象添加到当前选中项，如图9-95所示。

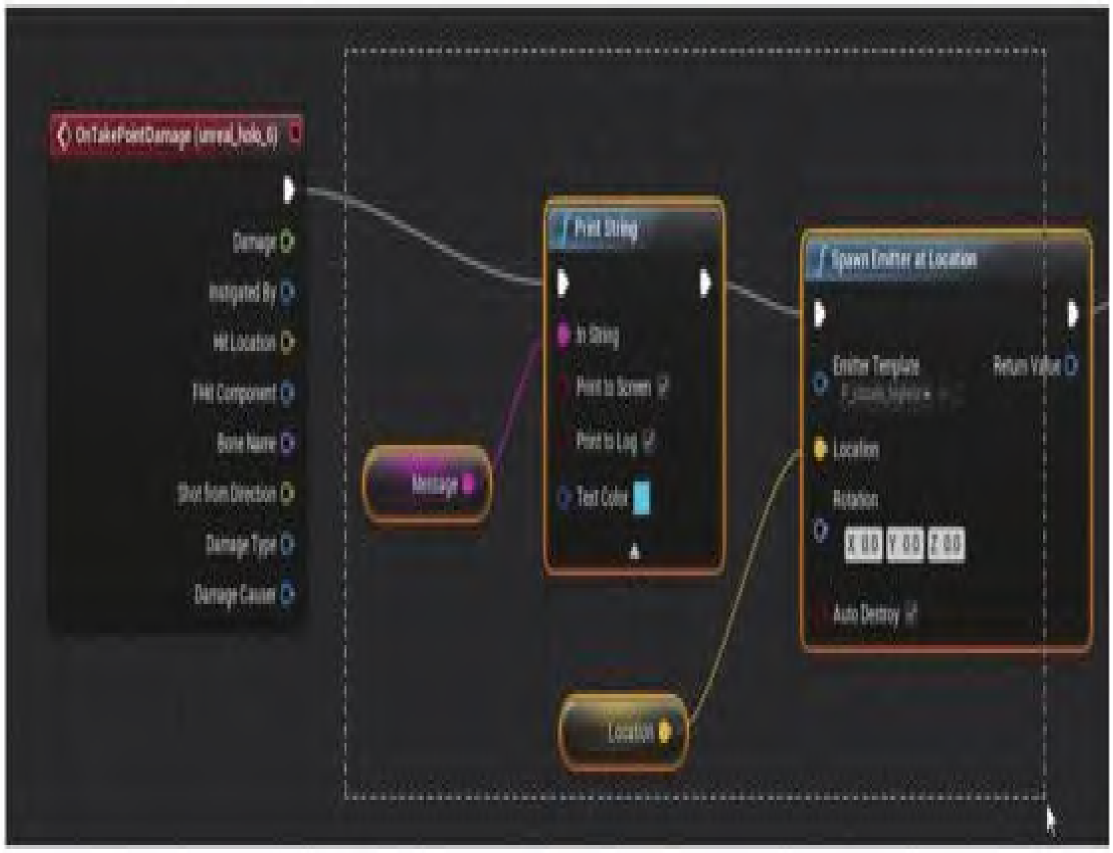



图9-95

要想取消选中所有节点，仅需单击图表编辑器选卡的空白区域即可。

## 9.6.5　移动节点

通过单击并拖动一个节点，可以移动该节点。如果选中了多个节点，那么单击选中项内的任何节点并拖动鼠标将移动所有节点，如图9–96所示。

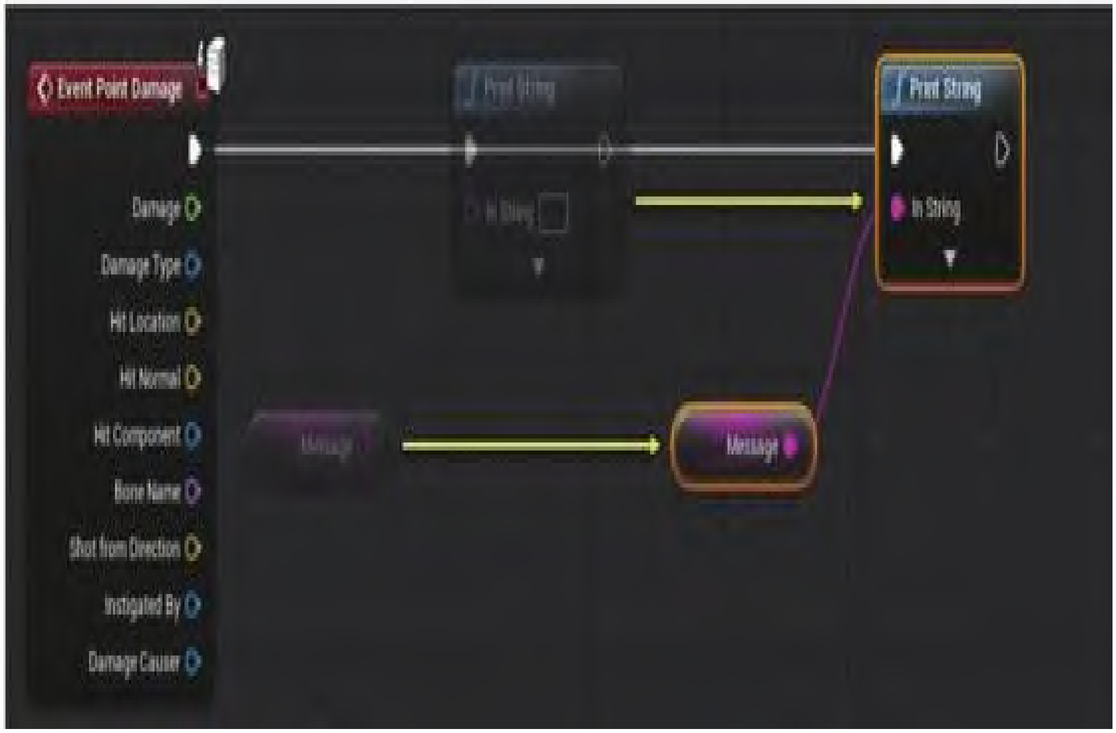



图9-96

## 9.6.6　引脚

节点两侧都可以有引脚。左侧的引脚是输入引脚，右侧的引脚是输出引脚，如图9-97所示。

有两种主要引脚类型：执行引脚（execution pins）和数据引脚（data pins）。

### 1.执行引脚

Execution pins（执行引脚）用于把节点连接到一起，来创建一个执行流。当激活一个输入执行引脚时，则执行该节点。一旦执行完一个节点，那么它将会激活一个输出执行引脚来继续执行流。执行引脚在没有连接时显示为一个引脚轮廓，当连接到另一个执行引脚时显示为实心引脚。函数调用（Function Call）节点仅有一个输入执行引脚和一个输出执行引脚，因为函数仅有一个入口点和一个出口点。其他类型的节点可以有多个输入和输出执行引脚，根据所激活的引脚的不同，可以产生不同的行为，如图9-98所示。

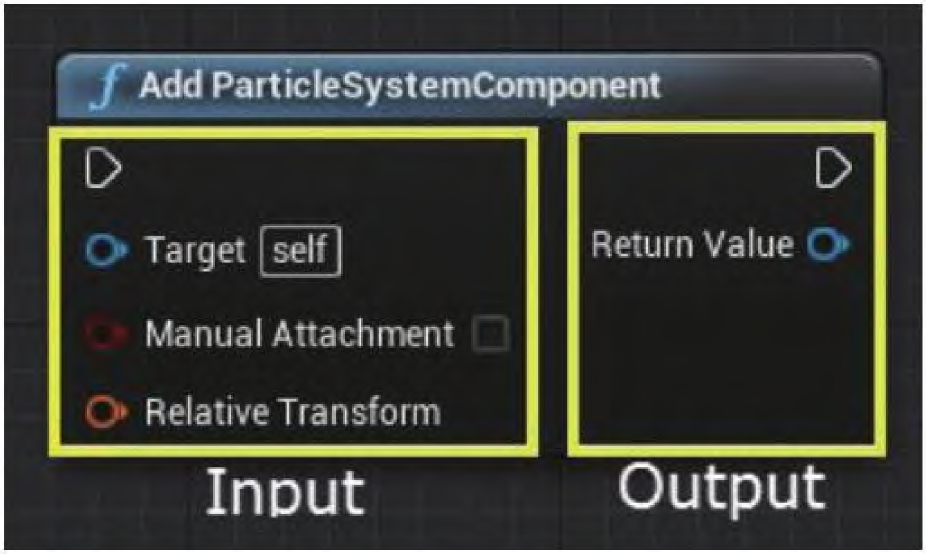



图9-97

图9-98

2.数据引脚

数据引脚被用来输入数据到节点或从节点输出数据。数据引脚为特定类型，可以与相同类型的变量相连接（这些变量也有其相应的数据引脚），或与另一节点的同类型数据引脚相连。和执行引脚一样，数据引脚在没有连接其他任何目标时，会显示为虚线，在连接后，会显示为实线，如图9-99所示。

节点可以有任意数量的输入或输出数据引脚。Function Call（函数调用）节点的数据引脚与相关函数的参数及返回值相对应，如图9-100所示。

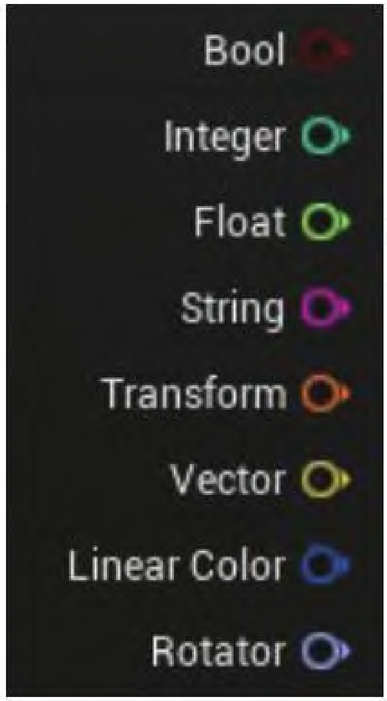



图9-99

图9-100

### 3. 自动类型转换

通过蓝图中的自动类型转换功能，不同数据类型的引脚可以相连接。当尝试在两个引脚间建立连接时，可以通过显示的工具提示信息识别兼容的类型，如图9-101所示。

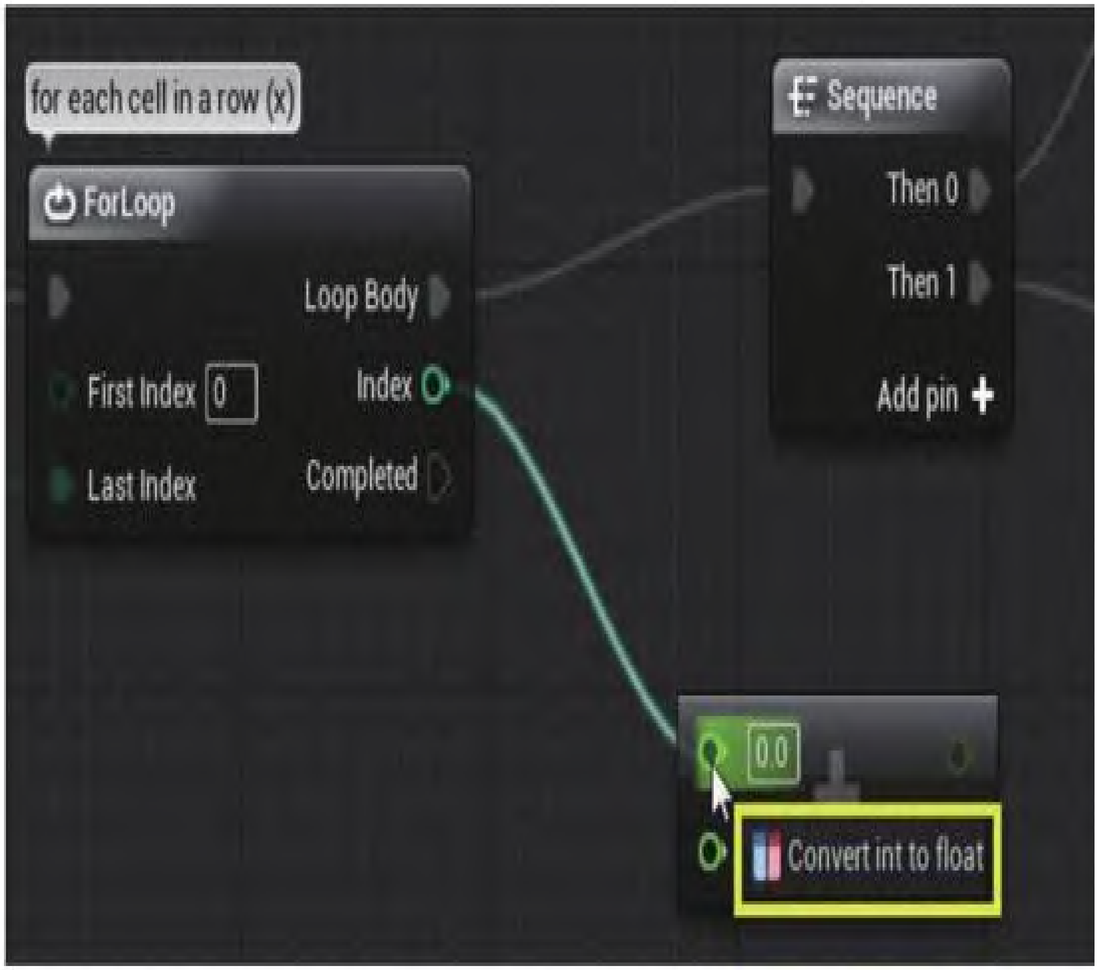



图9-101

从一种类型的引脚拖动一条线连接到另一种类型的引脚，但是这两种类型必须是兼容的，那么将创建一个Autocast（自动类型转换）节点连接两个引脚，如图9-102所示。

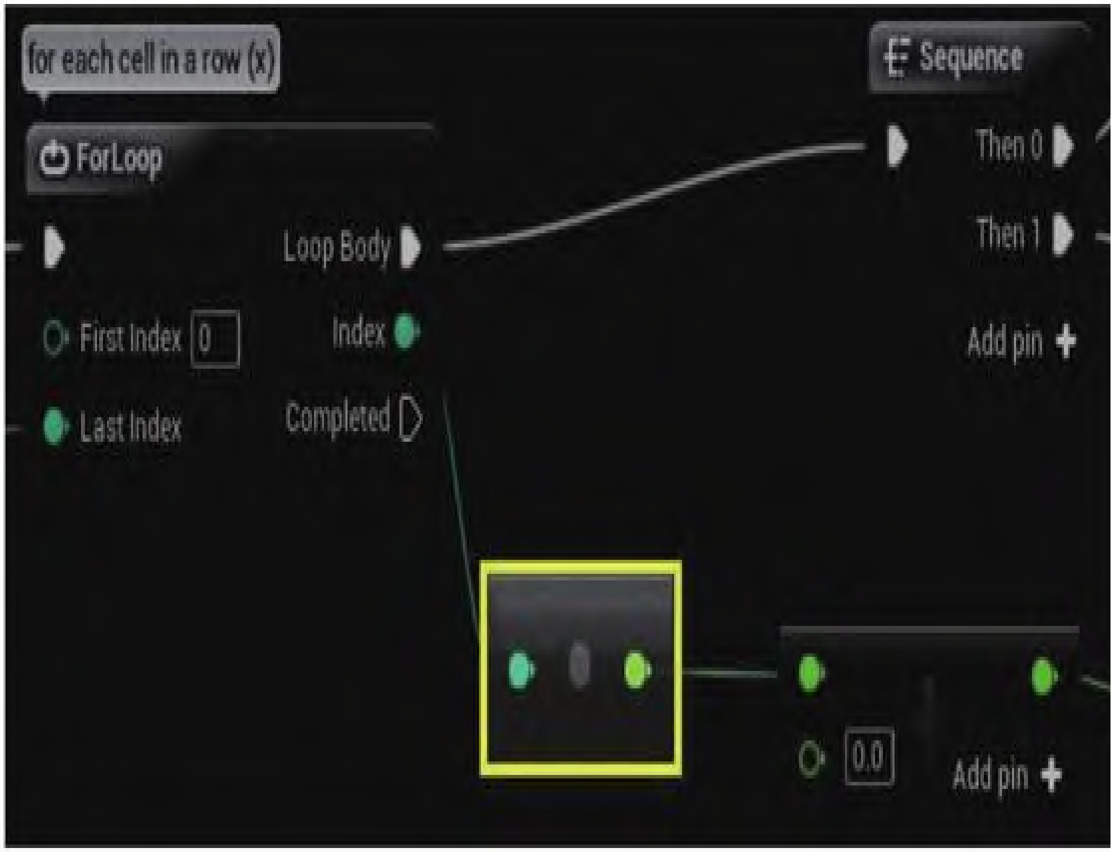



图9-102

Promote to Variable（提升为变量）

在蓝图中，数据引脚所表示的值可以通过Promote to Variable（提升为变量）命令转换为一个变量。这个命令会在蓝图中创建一个新的变量，并将其连接到那个提升为变量的数据引脚上。对于输出数据引脚来说，可以使用Set节点来设置新变量的值。从本质上讲，这仅是手动地添加一个新变量到图表中并将其和数据引脚相连的快捷方式。

你可以把一个节点上的任何输入或输出引脚提升为变量，除非该引脚是无类型的数组引脚，如图9-103所示。

**Step 1**　右击你想提升为变量的引脚，然后从弹出的菜单中选择Promote to Variable（提升为变量），如图9-104所示。

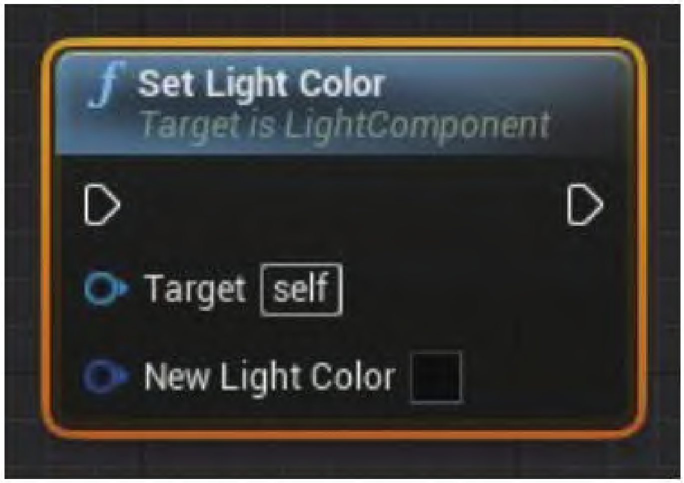



图9-103

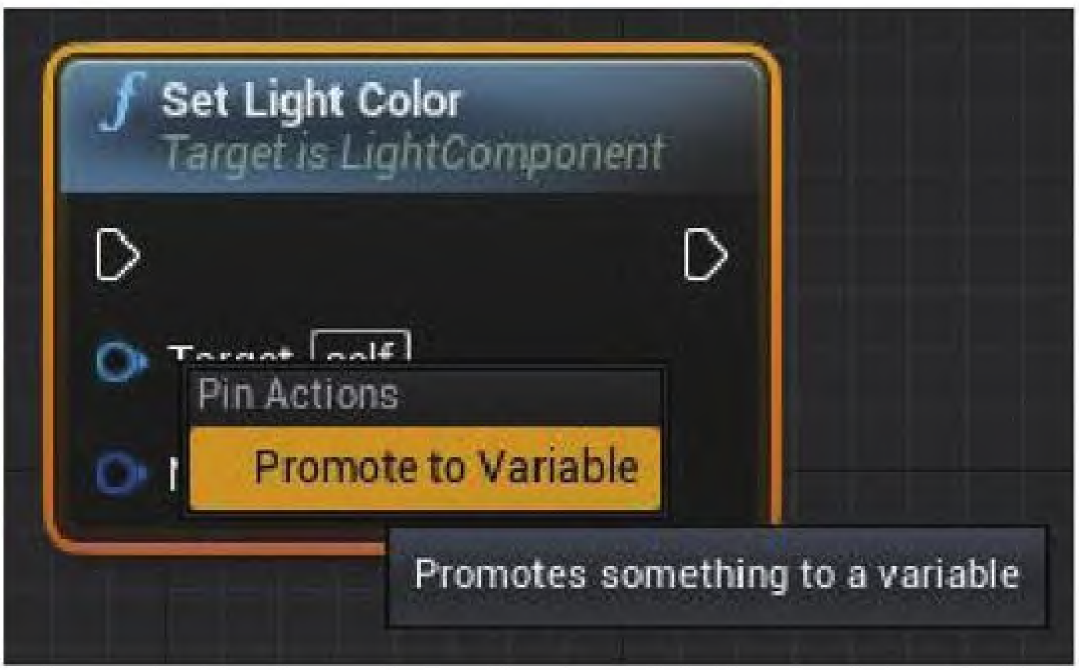



图9-104

**Step 2** 在新变量名称菜单中输入你的新变量的名称，如图9-105所示。

图9-105

图表上将会出现到那个新变量的引用。将一个输入引脚提升为变量将创建一个Get节点，将一个输出引脚提升为变量将创建一个Set节点，如图9-106所示。

图9-106

你的新变量将会列出在我的蓝图选卡中，在那里你可以选择它并在Details（详细信息）选卡中编辑它的详细信息。

## 9.6.7　连线

引脚之间的连接称为连线。连线代表执行流程或者数据流向。

### 1. 执行引脚连线

执行引脚间的连线代表执行的流程。执行连线显示为白色的箭头，箭头从一个输出执行引脚指向一个输入执行引脚。箭头的方向表明执行流程的走向，如图9-107所示。

图9-107

当正在执行“执行引脚”间的连线时，执行引脚连线会产生一个可视化的标识符。在运行过程中，当一个节点完成执行并激活了下一个节点时，执行引脚间的连线突出显示，表明正在从一个节点转移到另一个节点，如图9-108所示。

图9-108

执行连线的可视化标识符会随着时间逐渐消失，如图9-109所示。

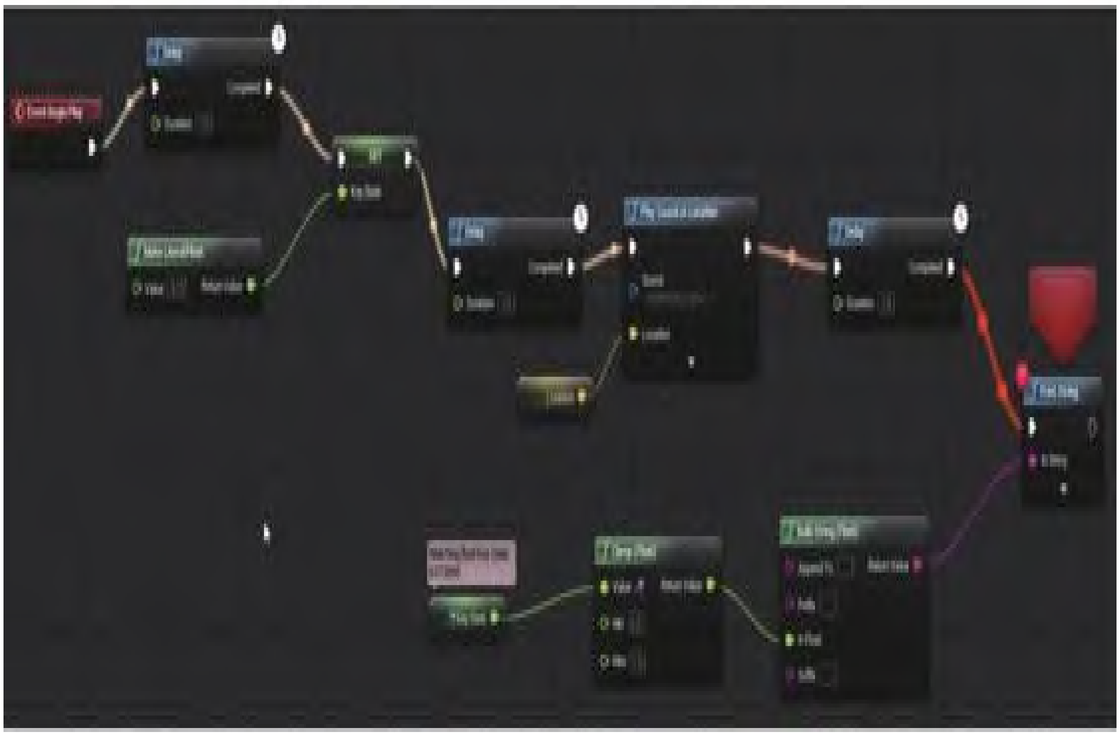


图9-109

## 2. 数据连线

数据连线把一个数据引脚连接到同种类型的另一个数据引脚上。数据连线显示为带颜色的箭头，用于可视化地表示数据的转移，箭头的方向代表数据移动的方向。和数据引脚的颜色一样，数据连线的颜色是由数据类型决定的，如图9-110所示。

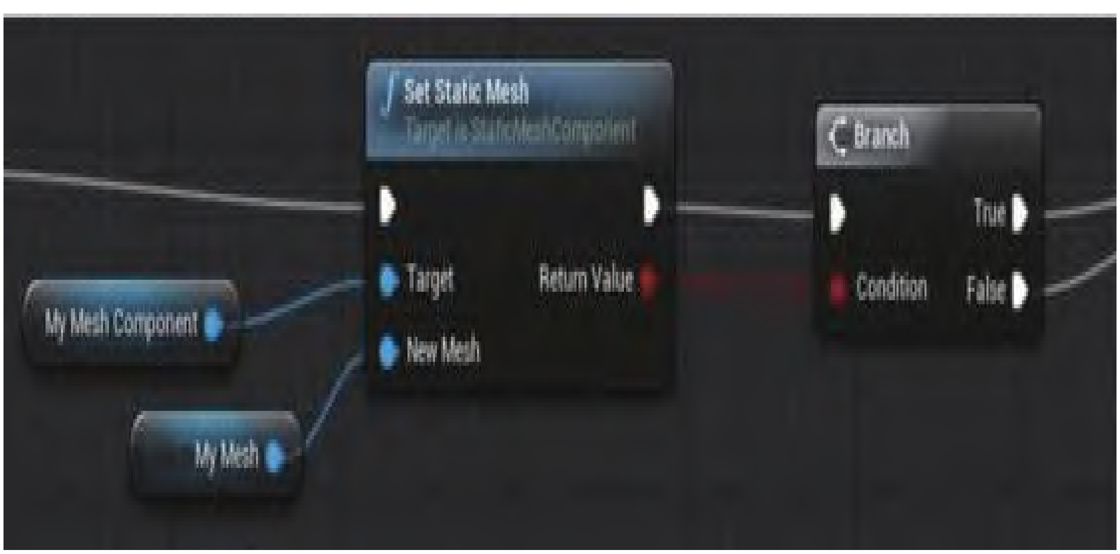



图9-110

3. 应用连线

在图标编辑器选卡中，可以使用以下几种方法之一创建连线。

•单击一个引脚并拖动鼠标，在同类型的另一个引脚上释放鼠标，这样来创建一个直接连接，如图9-111所示。

图9-111

仅能在两种兼容类型的引脚间创建连接。如果你向一个不兼容的引脚上拖动一个连接，将显示一个错误，提示你不能建立连接，如图9-112所示。

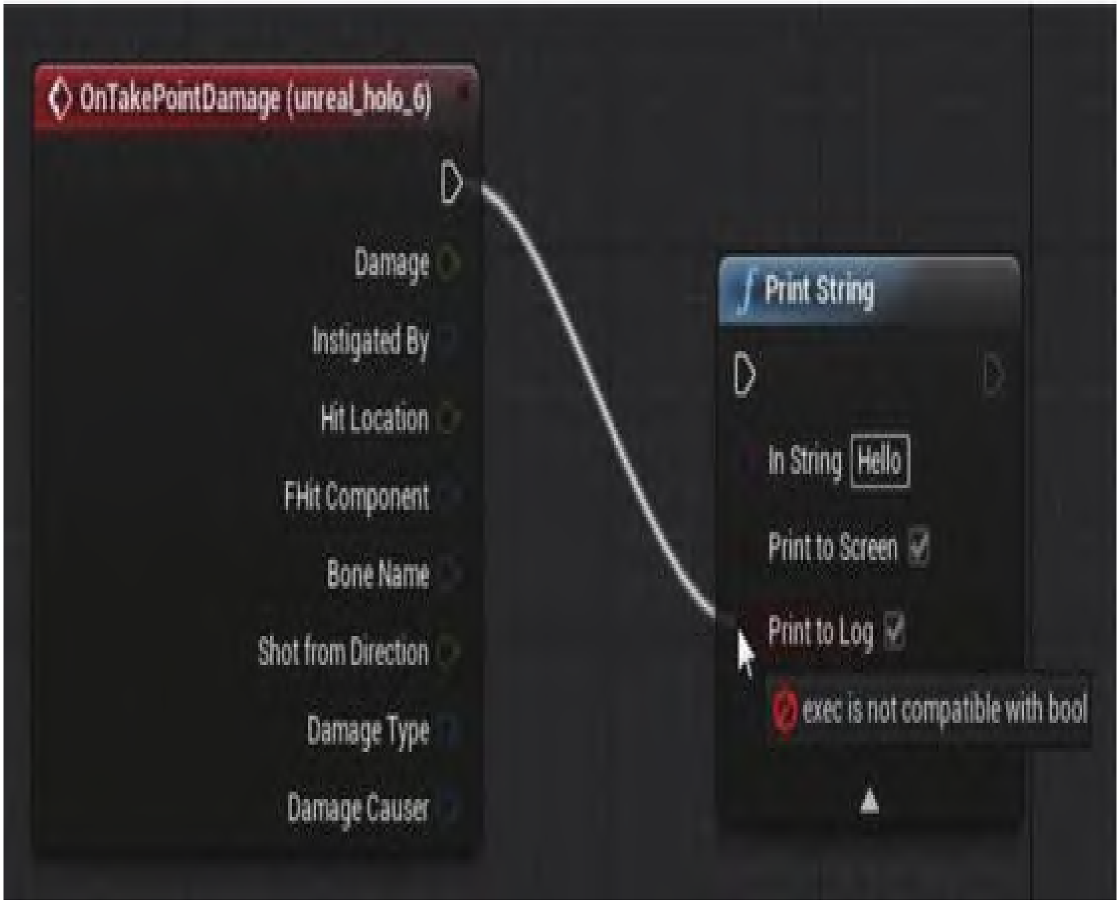



图9-112

•从一个引脚拖动一个连接并在空白区域释放鼠标，会调出一个情境关联的菜单，该菜单中列出了和连线起始节点类型相兼容的所有节点。从列表中选择一个节点将会创建那个节点的一个新实例并且连线会连接到该节点的一个兼容引脚上，如图9-113和9-114所示。

OnTakePointDamage (unreal_holo_6)
Place a new node.
Damage
Instigated By
Hit Location
FHit Component
Bone Name
Shot from Direction
Damage Type
Damage Causer

图9-113

图9-114

通过使用以下方法可以断开两个引脚间的连线：

•按Alt键+单击其中一个连接的引脚。

•右击所连接的其中一个引脚，并选择Break Link（断开连接），如图9-115所示。

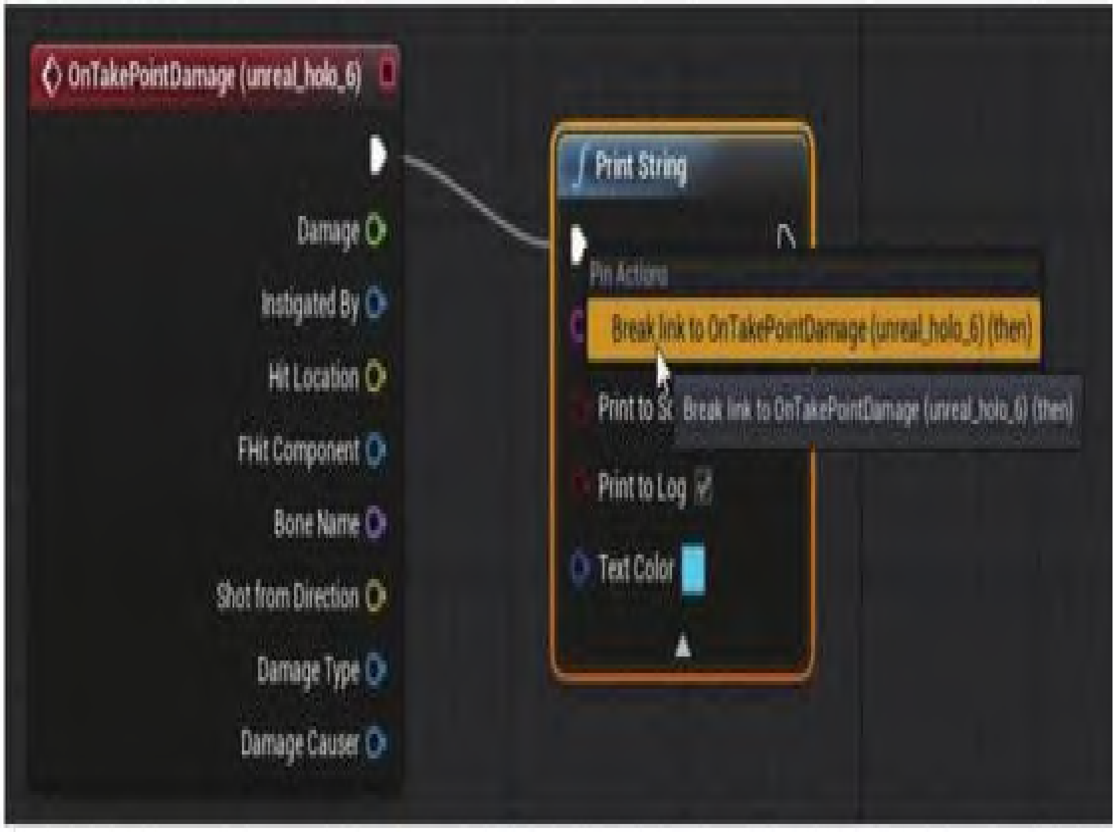



图9-115

## 9.6.8　合并的图表

出于组织管理的目的，图表中的一组节点可以合并到一个子图表中。这样会创建一个具有层次结构的图表，在父项图表中可以把一个大的或复杂的图表部分作为一个单独的节点看待，该节点具有输入和输出，但是仍然可以编辑合并的图表中的内容，如图9-116和图9-117所示。

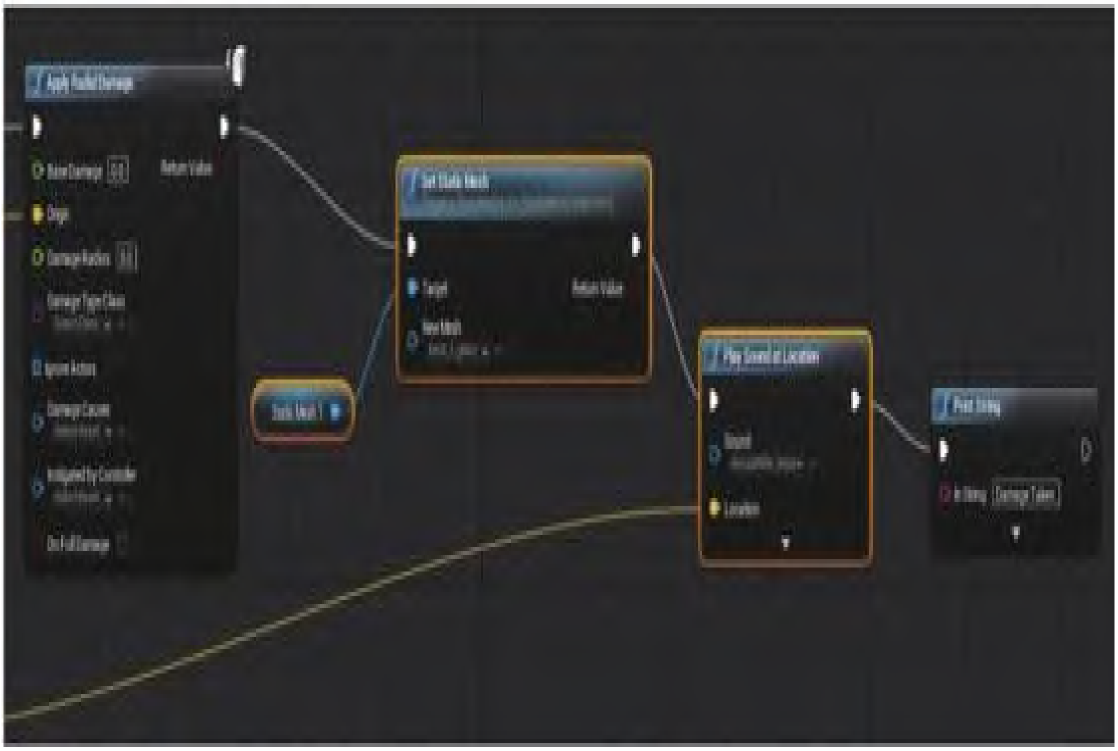


图9-116

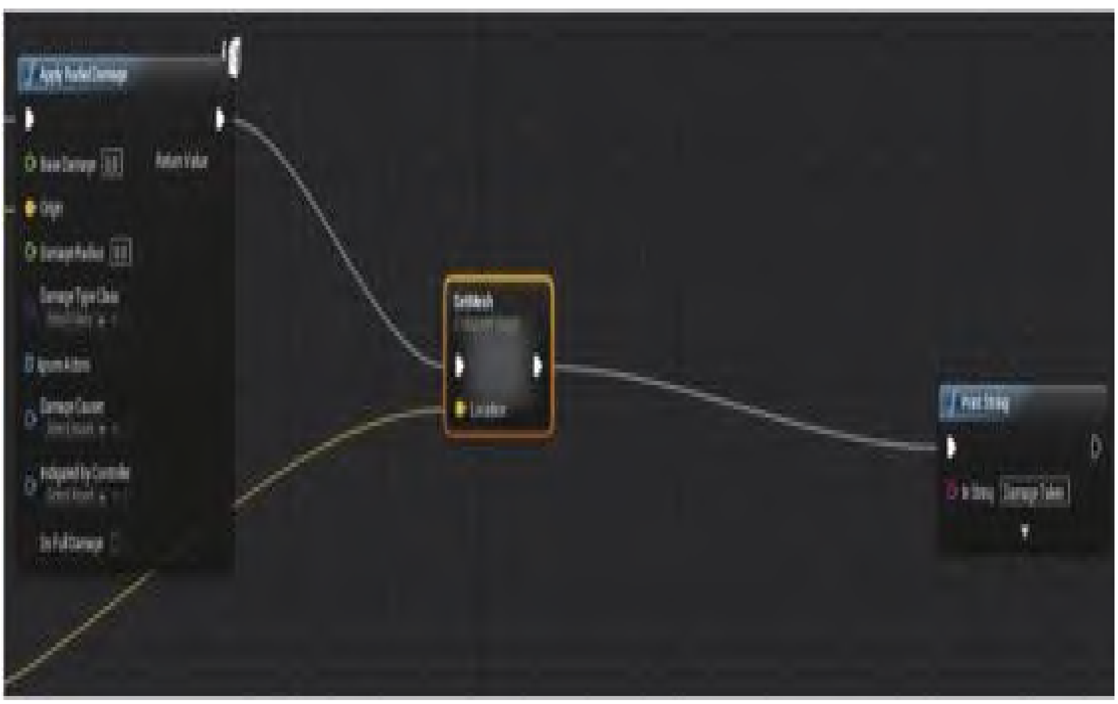


图9-117

和宏不同，一组合并的节点是不能共享的，即使在一个单独的关卡蓝图或类蓝图中也不可以共享。如果你复制此合并节点，它会复制内部图表。如

果你想创建同种近似行为的多个变种，这个操作是非常方便的，但是这也意味着任何缺陷修复都要应用到每个拷贝版本中。设计这个功能的主要目的是整理图表，隐藏内部复杂度，而不是用于共享或重用。

1.通道

合并的图表使用通道节点来和包含它的图表进行外部通信和交互，如图9-118所示。

图9-118

Inputs（输入）通道节点作为合并图标的入口点。它包含和父项图表中合并图表节点上的输入引脚相对应的执行引脚和数据引脚，如图9-119所示。

Outputs（输出）通道节点作为合并图标的出口点。它包含和父项序列中合并图表节点上的输出引脚相对应的执行引脚和数据引脚，如图9-120所示。

图9-119

图9-120

这些引脚是在合并节点时自动生成的。连接到序列中第一个节点的引脚上的任何执行引脚连线和数据引脚连线，都会导致在输入通道节点上创建对应的引脚，这些引脚在父项序列中的合并图表节点上作为输入引脚。类似地，任何连接到序列最后一个节点的执行引脚连线或数据引脚连线，都会导致在输出（Outputs）通道节点上创建对应的引脚，从而在父项序列中作为合并图表的引脚。

**2.合并一组节点**

**Step 1** 在图表中选择要合并的节点，这可以通过单击并在节点周围拖动一个区域选择框或者通过按住Ctrl键并单击每个节点来实现，如图9-121所示。

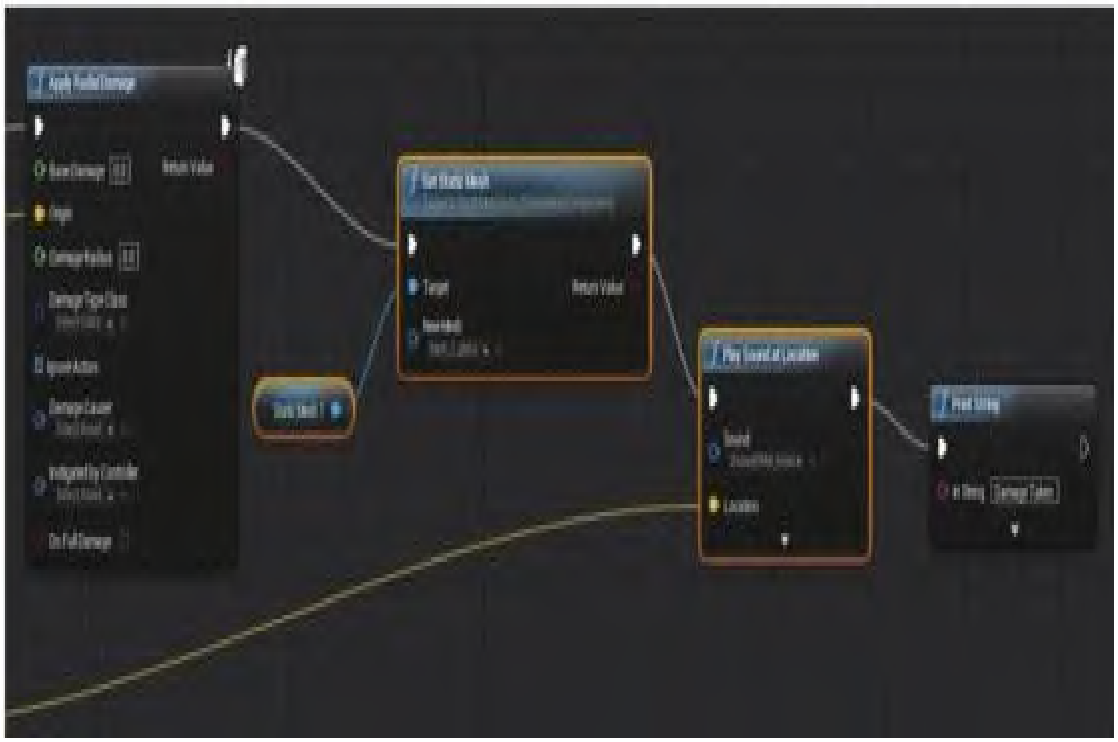



图9-121

**Step 2**　右击其中一个节点并选择Collapse Nodes（合并节点），如图9-122所示。

**Step 3**　在出现的新节点的文本域中输入该合并图标的名称并按下回车键，如图9-123所示。

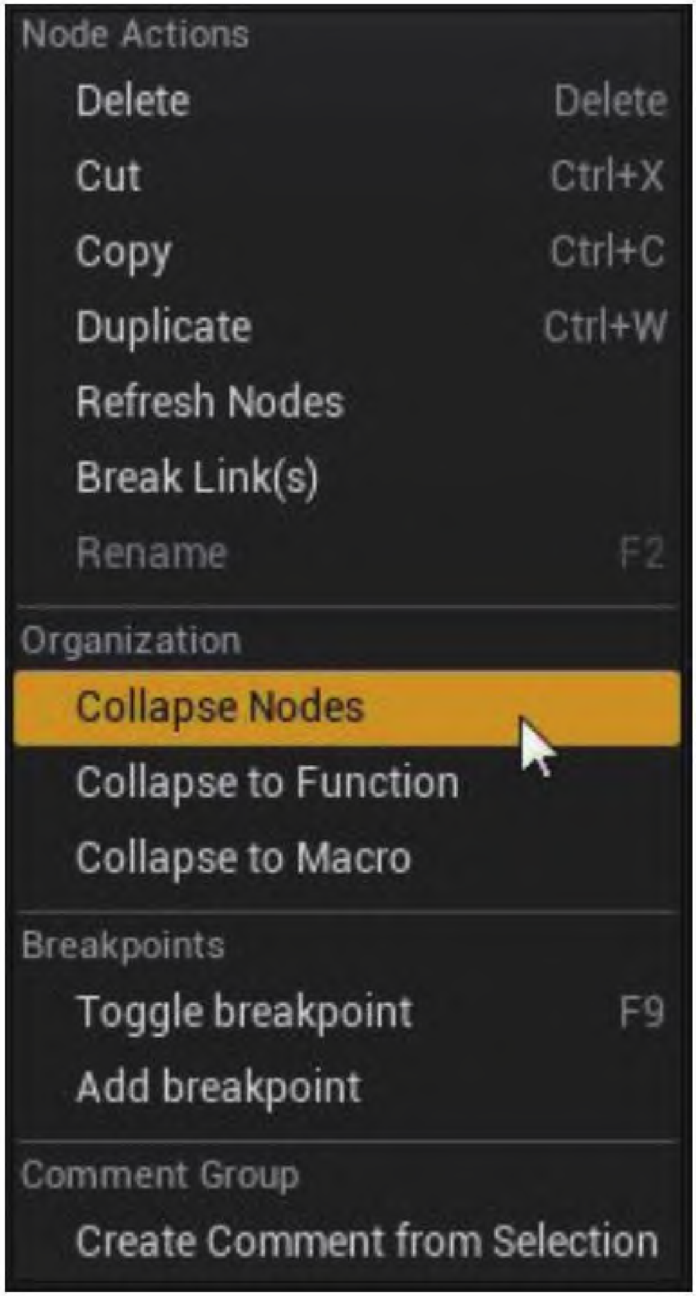

Node Actions
Delete
Delete
Cut
Ctrl+X
Copy
Ctrl+C
Duplicate
Ctrl+W
Refresh Nodes
Break Link(s)
Rename
F2
Organization
Collapse Nodes
Collapse to Function
Collapse to Macro
Breakpoints
Toggle breakpoint
F9
Add breakpoint
Comment Group
Create Comment from Selection

图9-122

图9-123

**Step 4** 现在，合并图表显示成了一个单独的节点，并在My Blueprint（我的蓝图）选卡中出现了到该合并图表的引用，如图9-124和图9-125所示。

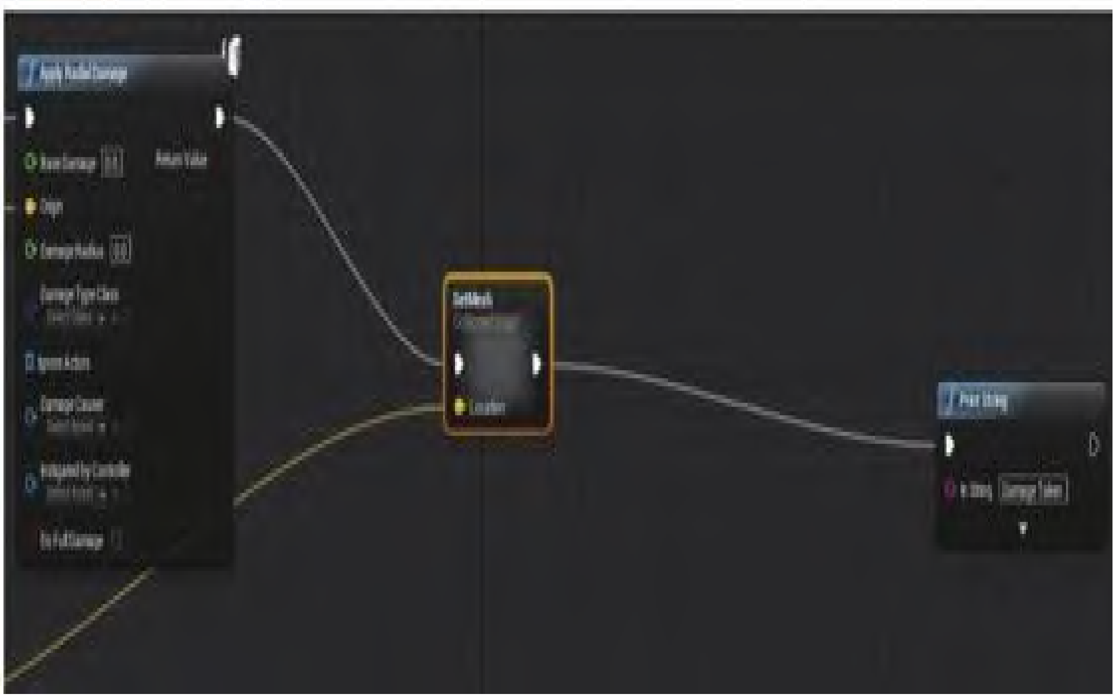


图9-124

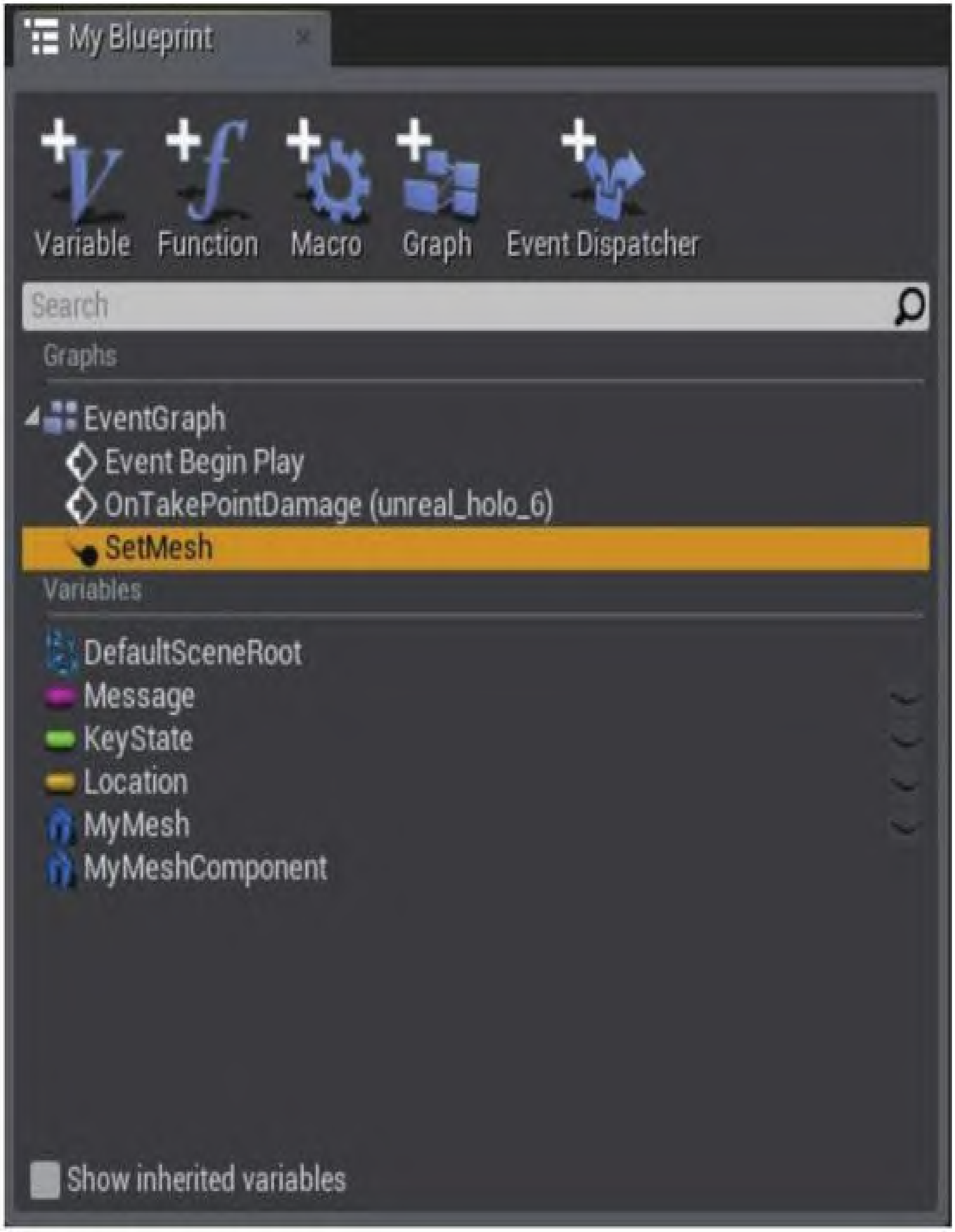



图9-125

**Step 5** 要想编辑该合并节点，请双击该合并图表节点或者在我的蓝图选卡中选择该子图表，如图9-126所示。

图9-126

### 3. 展开一个合并的图表

**Step 1**　右击一个合并图表节点，并选择Expand Node（展开节点），如图9-127所示。

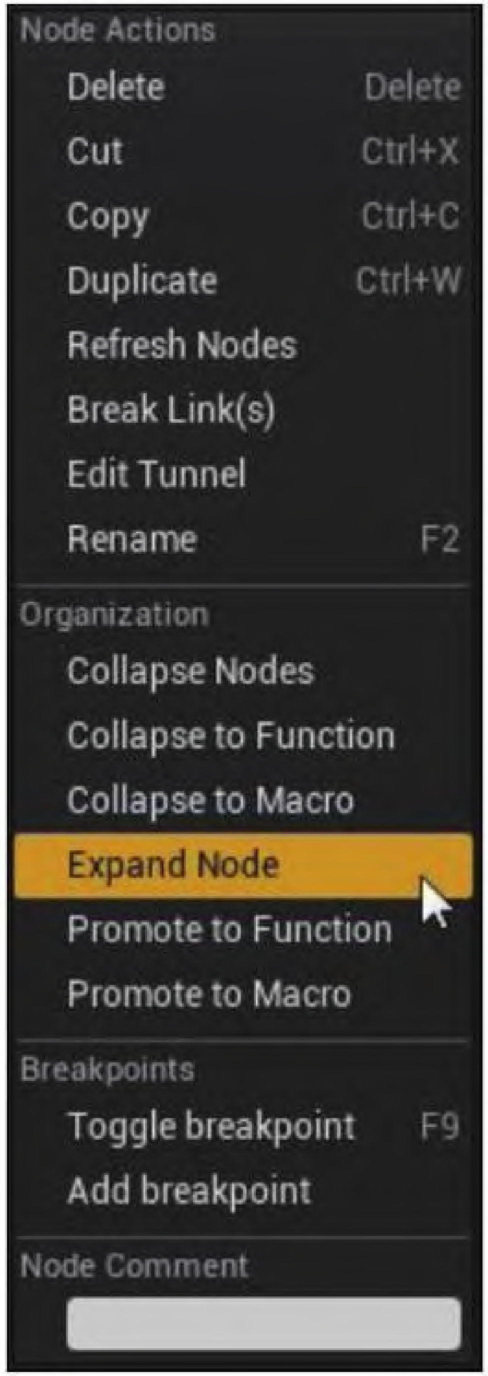



图9-127

**Step 2** 合并图表节点会被它所包含的节点所代替，不再出现在我的蓝图选卡的图表层次结构中，如图9-128所示。

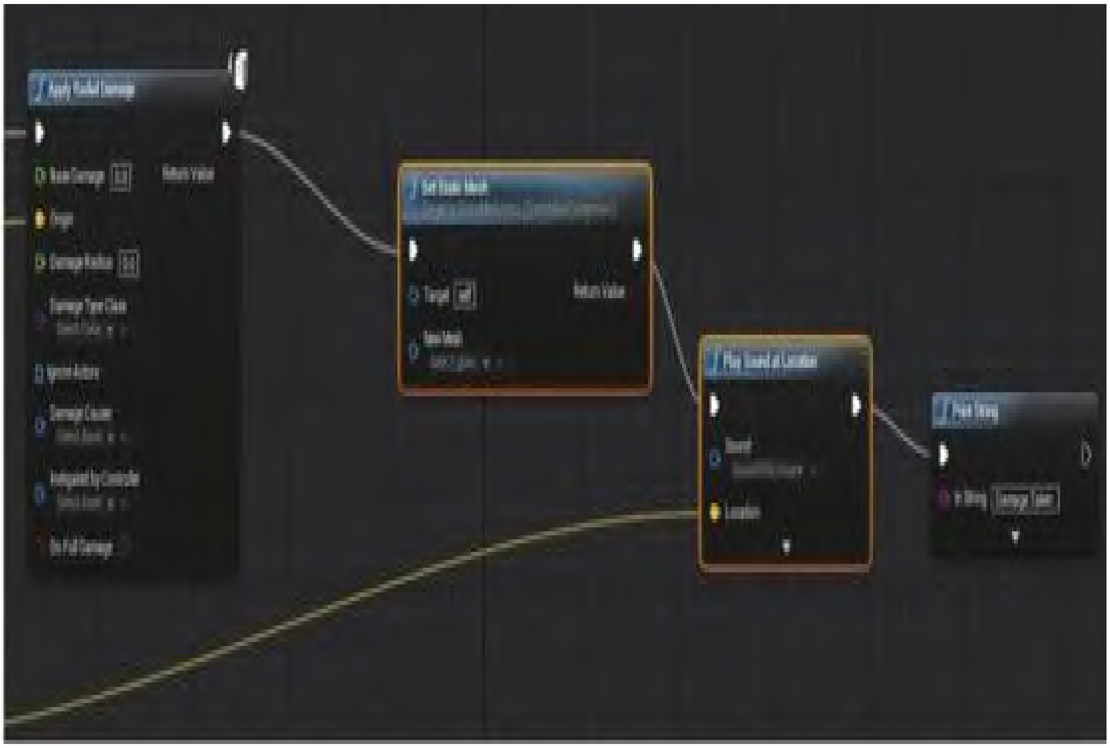


图9-128

9.7

# 随机流

RandomStreams（随机流）允许在蓝图、关卡蓝图及针对动画的动画蓝图中重复地生成及应用随机数。当设置类似于散射物体或者构建程序化的场景时，这是非常有用的。此时你可能需要一种随机的效果，但是同时又想确保每次计算蓝图时产生一致的分布。之前，使用随机值会导致每次计算蓝图时产生不同的分布，这意味着当移动蓝图或者执行其它的导致需要重新计算图表的动作时，会产生完全不同的效果。通过使用随机流，你可以基于一个种子值调整效果来获得期望的结果，然后在维持整体效果的过程中执行任何其他修改。

## 9.7.1 RandomStream（随机流）变量

RandomStreams（随机流）在蓝图中以一种特殊类型的结构体变量呈现。像其他变量一样，你可以通过在图表模式中的我的蓝图面板中创建此类变量并添加它们，如图9-129所示。

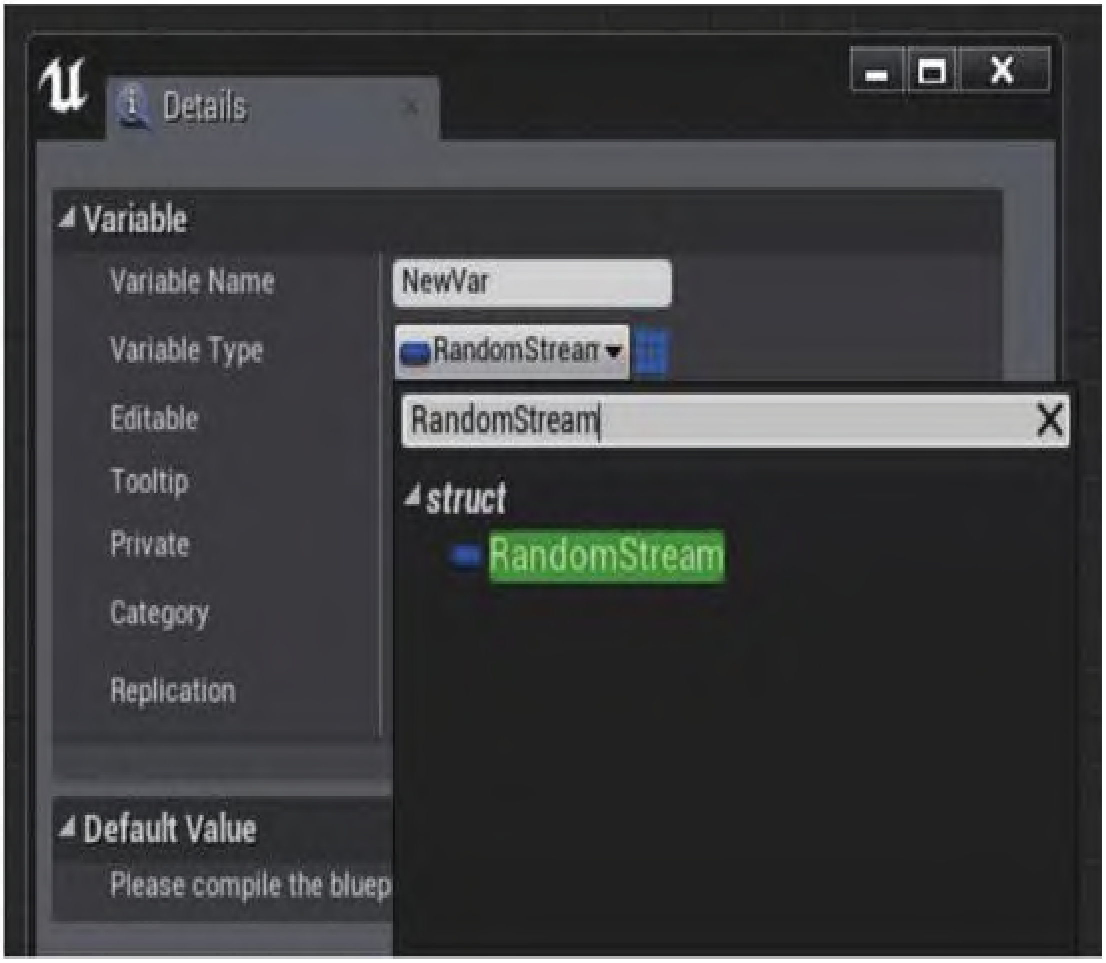



图9-129

## 9.7.2　随机流函数

为了使用随机流变量，我们提供了一组函数，它们可以取入一个RandomStream（随机流）作为输入，且根据函数功能不同它们输出一个不同类型的随机值，见图9-130和下表。

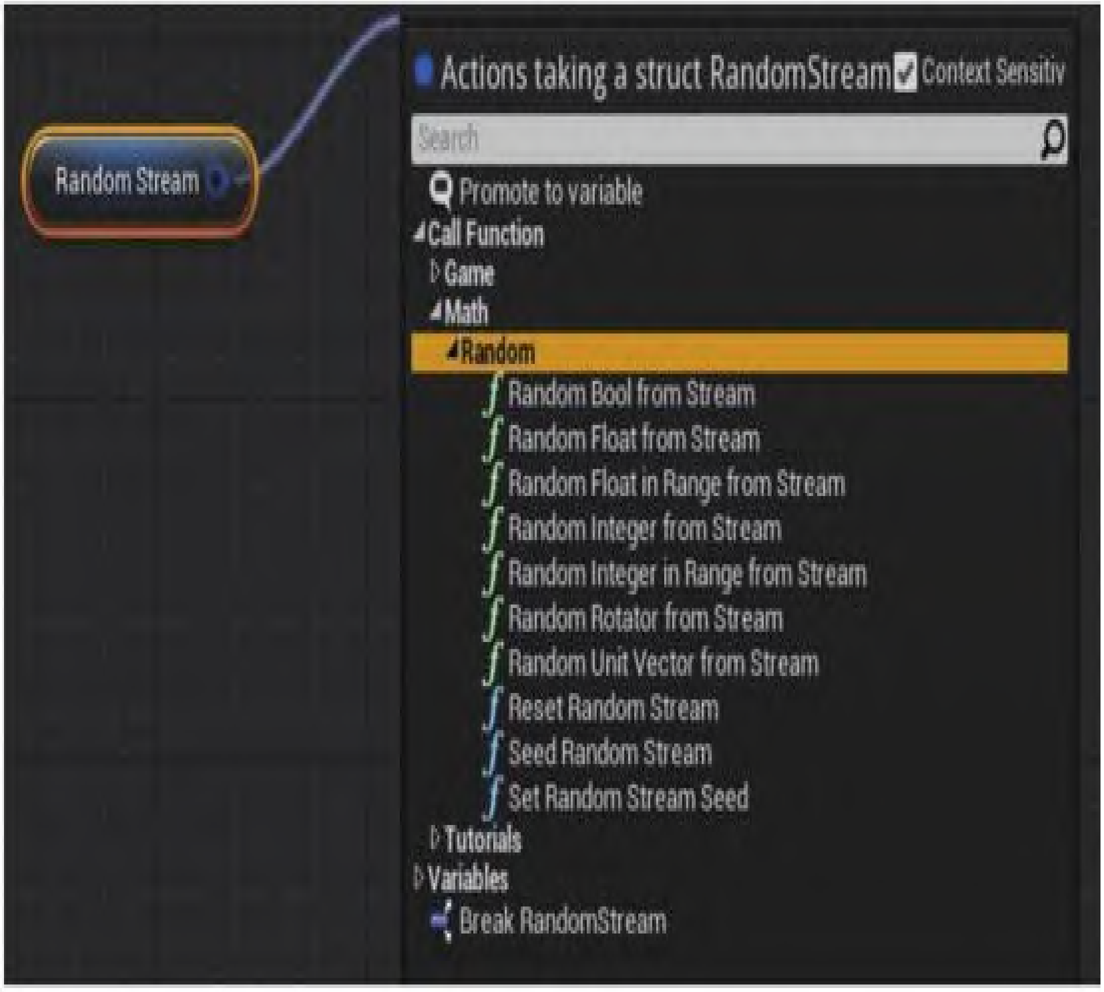

Random Stream
Actions taking a struct RandomStream
Context Sensitiv
Search
Promote to variable
Call Function
Game
Math
Random
Random Bool from Stream
Random Float from Stream
Random Float in Range from Stream
Random Integer from Stream
Random Integer in Range from Stream
Random Rotator from Stream
Random Unit Vector from Stream
Reset Random Stream
Seed Random Stream
Set Random Stream Seed
Tutorials
Variables
Break RandomStream

图9-130

| 函数 | 描述 |
| --- | --- |
| Random Bool from Stream | 随机输出 0 或 1 |
| Random Float from Stream | 随机输出 (0.0,1.0) 范围之间的一个浮点值 |
| Random Float in Range from Stream | 随机输出 (Min,Max) 范围之间的一个浮点值 |
| Random Integer From Stream | 输出 (0,Max-1) 范围之间的一个均匀分布的随机整型值 |
| Random Integer In Range From Stream | 输出 (Min,Max) 范围之间的一个随机整型值 |
| Random Rotator From Stream | 输出一个随机的 Rotator（旋转度）值 |
| Random Unit Vector From Stream | 输出一个随机的单位长度的向量值 |

选择上面显示的其中一个函数，将把该函数放置到图表中并连接到RandomStream变量上，如图9-131所示。

图9-131

以下示例展示了如何使用RandomStream（随机流）函数来随机地放置及旋转一个属于某个蓝图的组件，如图9-132所示。

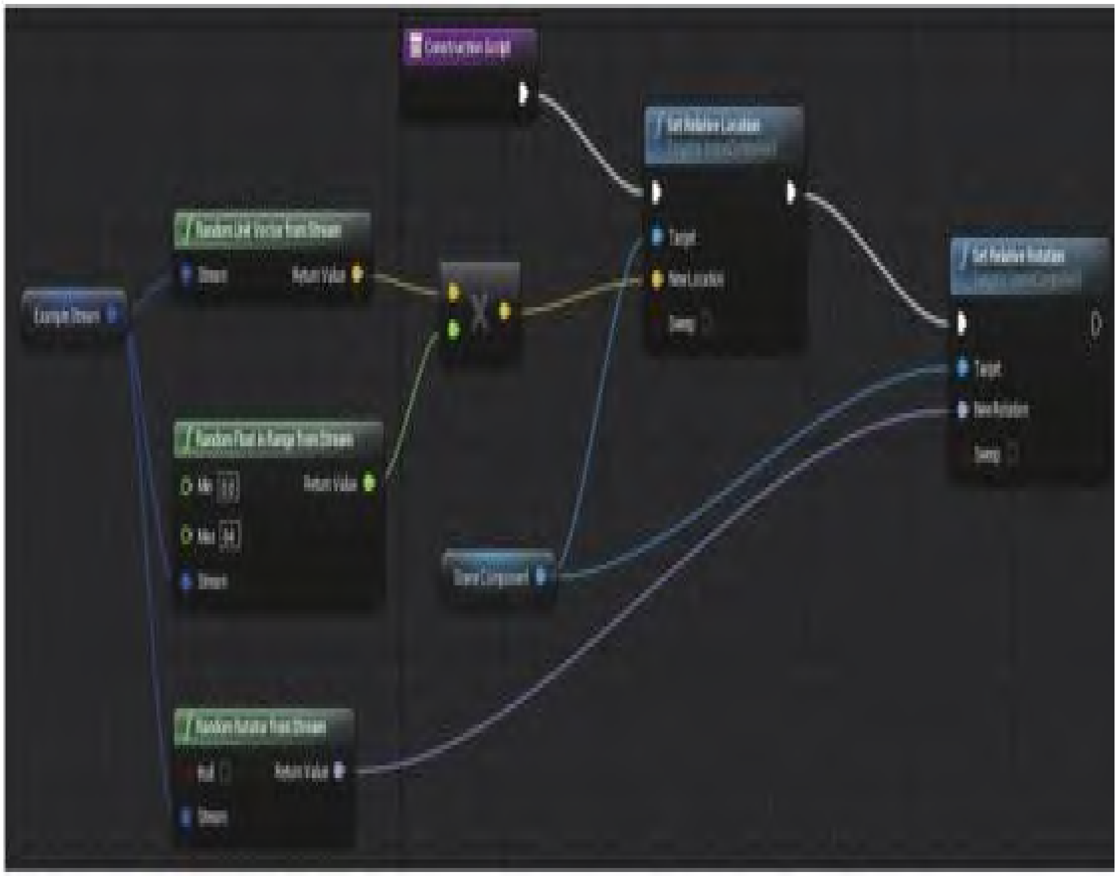


图9-132

正如你看到的，随机的布尔值、浮点型值、整型值、向量及旋转度都可以从同一个流中提取出来。

## 9.7.3　初始种子

Initial Seed（初始种子）属性用于计算随机值流。每次计算一个单独的随机种子所产生的随机值序列都将是一样的，这为我们提供了前面提到的一致性。不同的种子生成不同的值序列。

所以，修改一个RandomStream的Initial Seed（初始种子）将会导致所生成的值发生变化。这可以用于调整一种随机效果，直到你获得你需要的序列或分布为止。蓝图的每个实例都会生成一个新的Initial Seed（初始种子值）。这意味着每次在世界中放置一个蓝图的实例时或者复制一个现有实例时，将赋予属于该蓝图的每个RandomStream变量一个新的Initial Seed（初始种子）值。所以，每个实例都将是不同的，并且可以对其进行调整来创建实际期望的效果。

要想直接在变量上设置Initial Seed（初始种子）属性，则RandomStream变量必须暴露为可编辑的。一旦暴露了该变量，那么当选中一个蓝图的实例时，就可以在详细信息选卡中看到此Initial Seed（初始种子）属性，如图9-133所示。

图9-133

在蓝图图表中，你还可以将Initial Seed（初始种子）设置为一个特定的值或一个新的随机值。

当你具有一个放置了很多个对象（比如草叶或石头）的蓝图，但你总是想以相同的方式放置它们，以便用于测试或其他目的时，Set Random Stream Seed节点就很有用了。它将覆盖将该蓝图放置到关卡中时随机设置的Initial Seed（初始种子）。

9.8

# 时间轴

Timeline nodes（时间轴节点）是蓝图中的特殊节点，它使你可以快速地设计基于时间的简单动画，并基于游戏中的事件进行播放。时间轴有点像简单的Matinee序列，因为它们都可以使得简单的值从中产生动画，且可以随着时间变化来激活事件。通过在图表选卡或My Blueprint（我的蓝图）选卡中双击该时间轴，你可以直接在蓝图编辑器中编辑这些节点。这些节点专门用来处理简单的、非过场动画式的任务，比如开门、改变光源或对场景中的Actor执行其他基于时间的操作，如图9-134所示。

输入和输出，如图9-135所示。

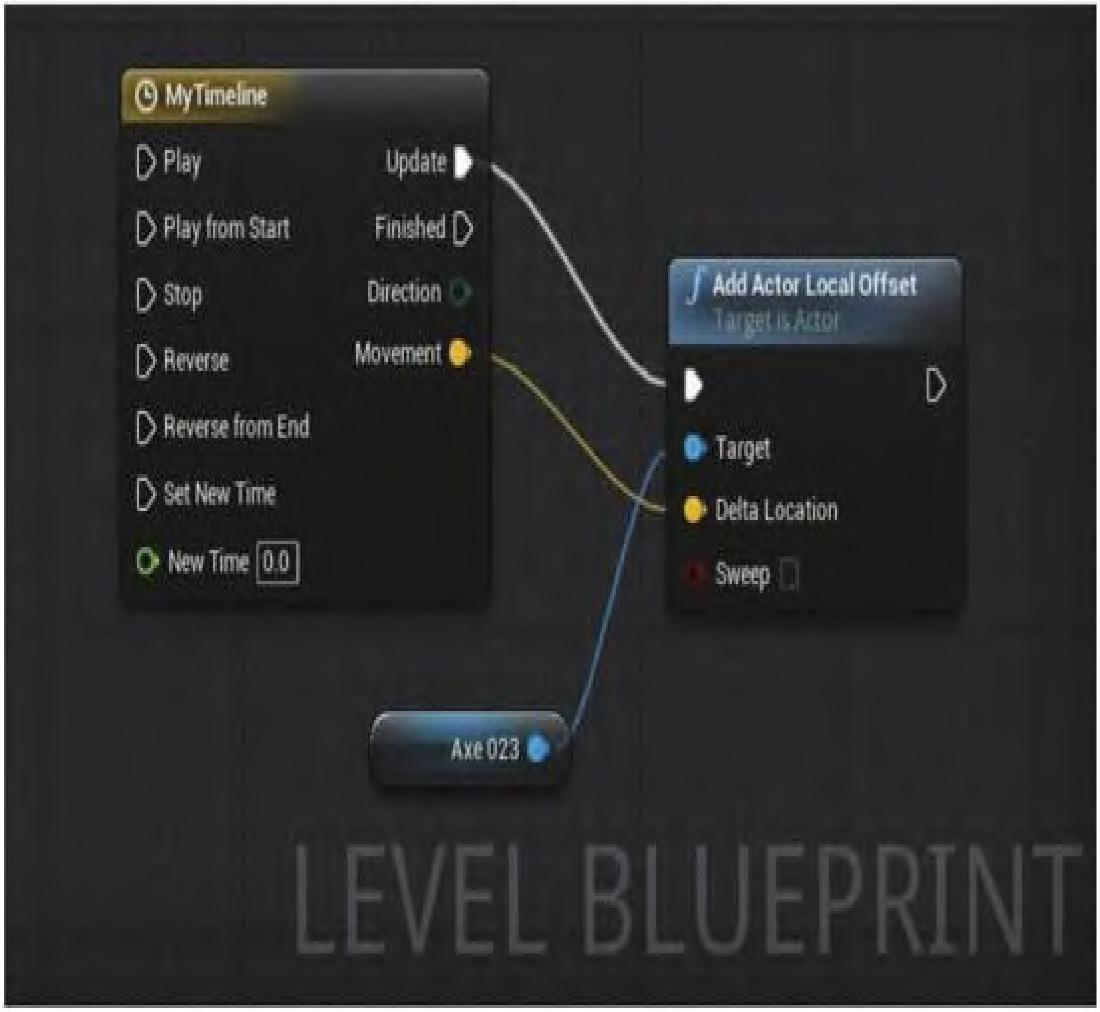

MyTimeline
Play
Update
Play from Start
Finished
Stop
Direction
Reverse
Movement
Reverse from End
Set New Time
New Time
0.0
Add Actor Local Offset
Target is Actor
Target
Delta Location
Sweep
Axe 023
LEVEL BLUEPRINT

图9-134

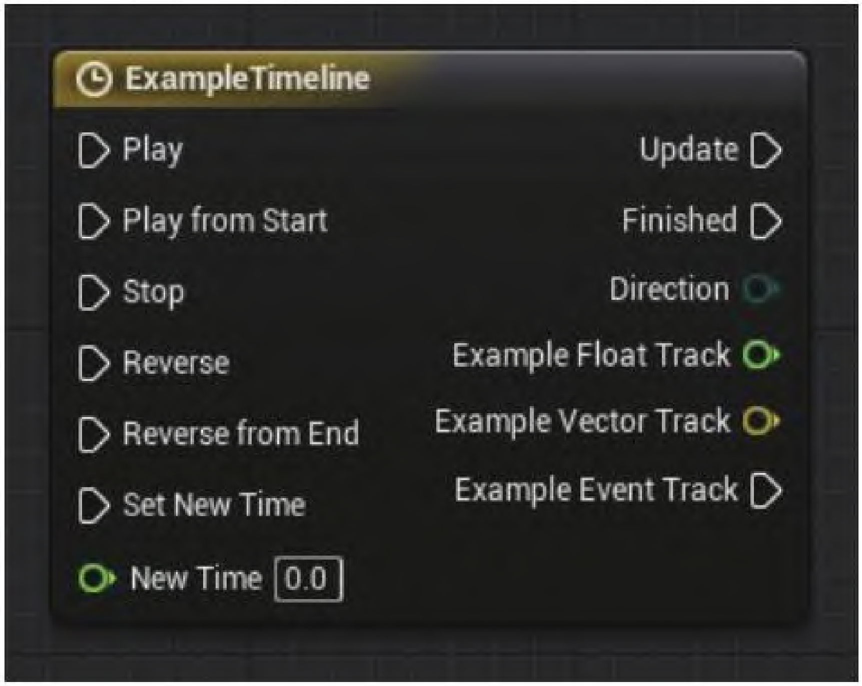



图9-135

时间轴本身具有以下输入和输出引脚。

| 项　目 | 描　述 |
| --- | --- |
| 输入引脚 | |
| Play（播放） | 使得时间轴从当前时间处开始正向播放 |
| Play from Start（从开始处播放） | 使得时间轴从开始处正向播放 |
| Stop（暂停） | 在当前时间处停止播放时间轴 |
| Reverse（反向播放） | 从当前时间处反向播放时间轴 |
| Reverse from End（从结尾处开始反向播放） | 从头开始反向播放时间轴 |
| Set New Time（设置新时间） | 将当前时间设置为 New Time（新时间）输入中的设置的变量（或输入） |
| New Time（新时间） | 该数据引脚取入一个代表时间的浮点值，以秒为单位，当调用 Set New Time（设置新时间）输入时，时间轴可以跳转到该浮点值设置的时间处 |

续表

| 项　目 | 描　述 |
| --- | --- |
| 输出引脚 | |
| Update（更新） | 一调用该时间轴就输出一个执行信号 |
| Finished（完成） | 当播放结束时输出一个执行信号，该引脚不会被 Stop 函数触发 |
| Direction（方向） | 输出枚举数据，指明了时间轴的当前播放方向 |

时间轴可以具有多个额外的输入数据引脚，来反映在时间轴中创建的轨迹类型。这些轨迹类型包括Float（浮点型）、Vector（向量型）和Event（事件）轨迹。

## 9.8.1　创建时间轴

**1. 概述**

本文涵盖了对于如何在蓝图中创建时间轴节点的概述。

**2. 创建时间轴**

**Step 1**　在蓝图中创建时间轴非常简单。在图表选项卡中右击，并从关联菜单中选择Add Timeline（添加时间轴），如图9-136所示。

**Step 2**　新的时间轴节点将会被添加到图表中，如图9-137所示。

在时间轴被添加后，你将可以看到其位于My Blueprint（我的蓝图）标签栏，如图9-138所示。

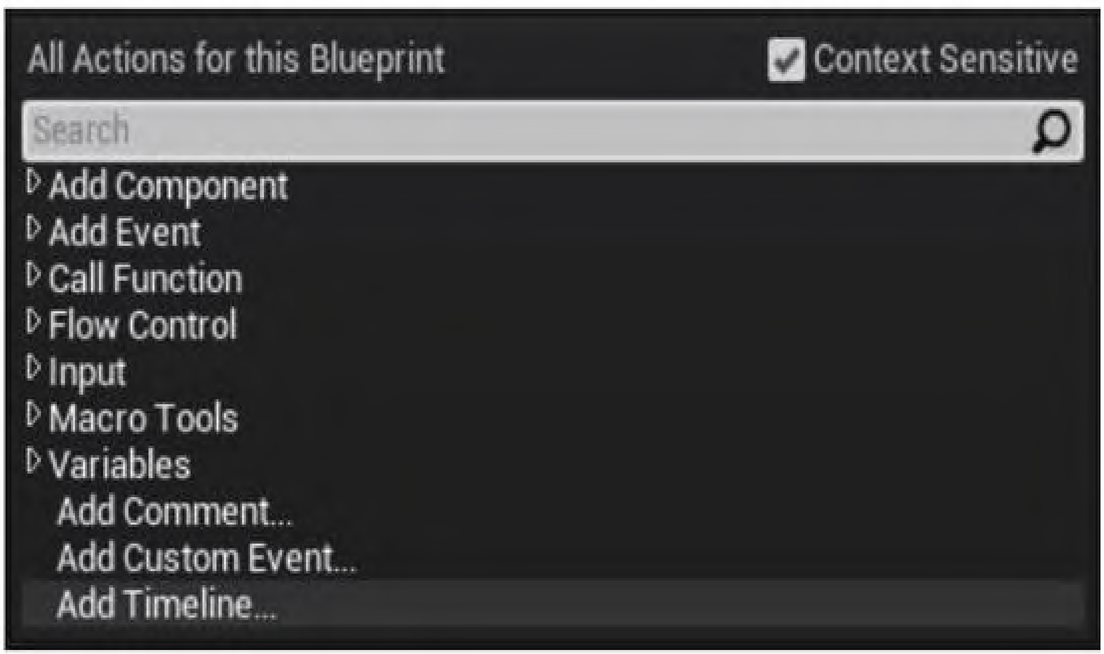



图9-136

Timeline_0
Play
Update
PlayFromStart
Finished
Stop
Direction
Reverse
ReverseFromEnd
SetNewTime
NewTime
0.0

图9-137

图9-138

### 3. 时间轴变量

在你创建了时间轴后，你可以在My Blueprint（我的蓝图）标签栏中注意到它是一个变量并可用。这样将可以引用时间轴组件，在需要使用时间轴节点时特别有用。这些节点使得我们可以访问时间轴的特定功能，比如时间轴的播放速度，如图9-139所示。

在本示例中，我们使用获取Timeline 0变量来查找该时间轴当前的Play Rate（播放速度）值，如图9-140所示。

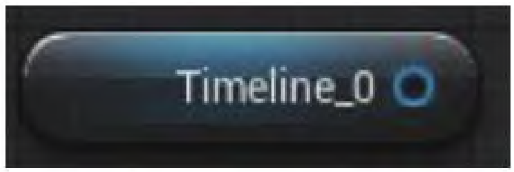



图9-139

图9-140

此类节点很多，可以让你查找时间轴节点的值并控制其行为。

### 4.重命名时间轴

当你在蓝图中使用多个时间轴时，很重要的一点是对每个时间轴进行相应命名，默认所有时间轴都采用“Timeline_X”的格式来命名（X为序列数字）。你可以通过在Graph（图表）选项卡内通过右击来重命名时间轴，或者你可以在My Blueprint（我的蓝图）选项卡内选择Rename（重命名），如图9-141和图9-142所示。

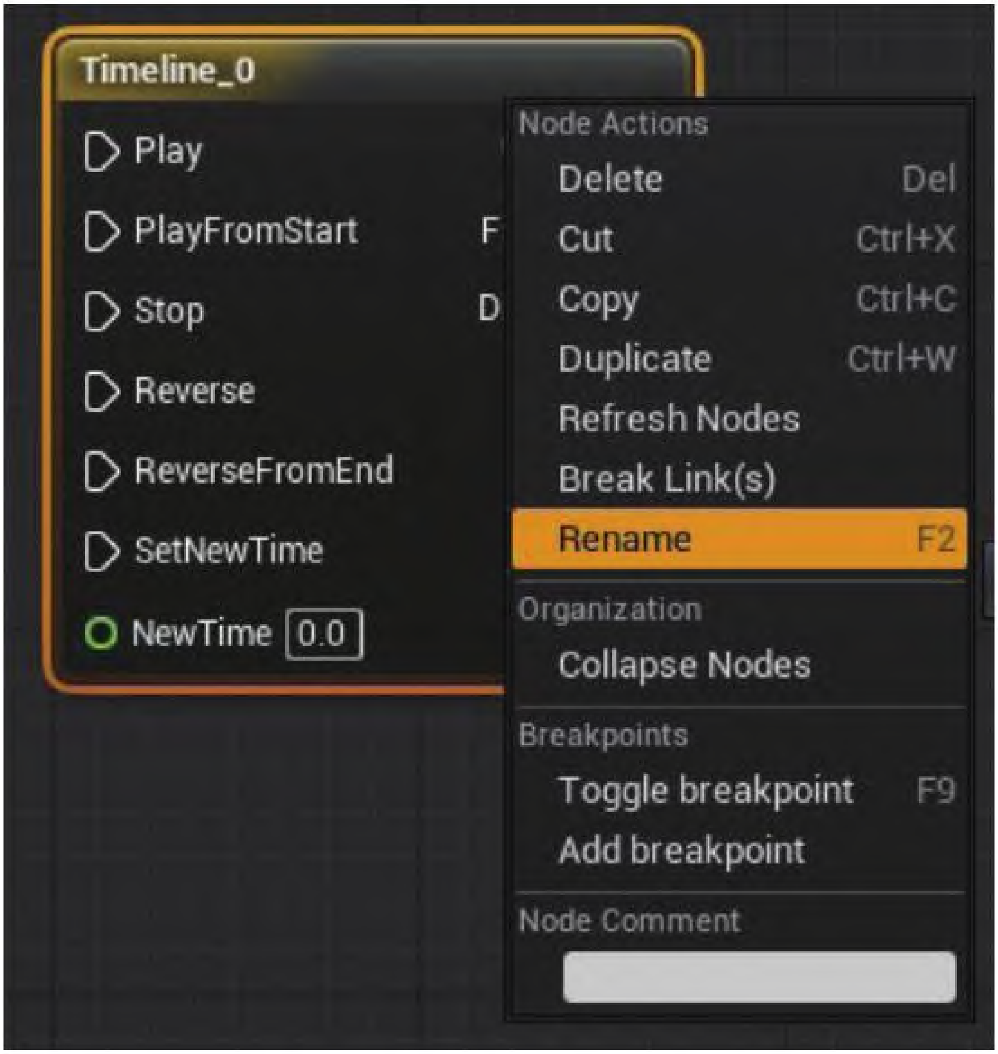

Timeline_0
Play
PlayFromStart
F
Stop
D
Reverse
ReverseFromEnd
SetNewTime
NewTime
0.0
Node Actions
Delete
Del
Cut
Ctrl+X
Copy
Ctrl+C
Duplicate
Ctrl+W
Refresh Nodes
Break Link(s)
Rename
F2
Organization
Collapse Nodes
Breakpoints
Toggle breakpoint
F9
Add breakpoint
Node Comment

图9-141

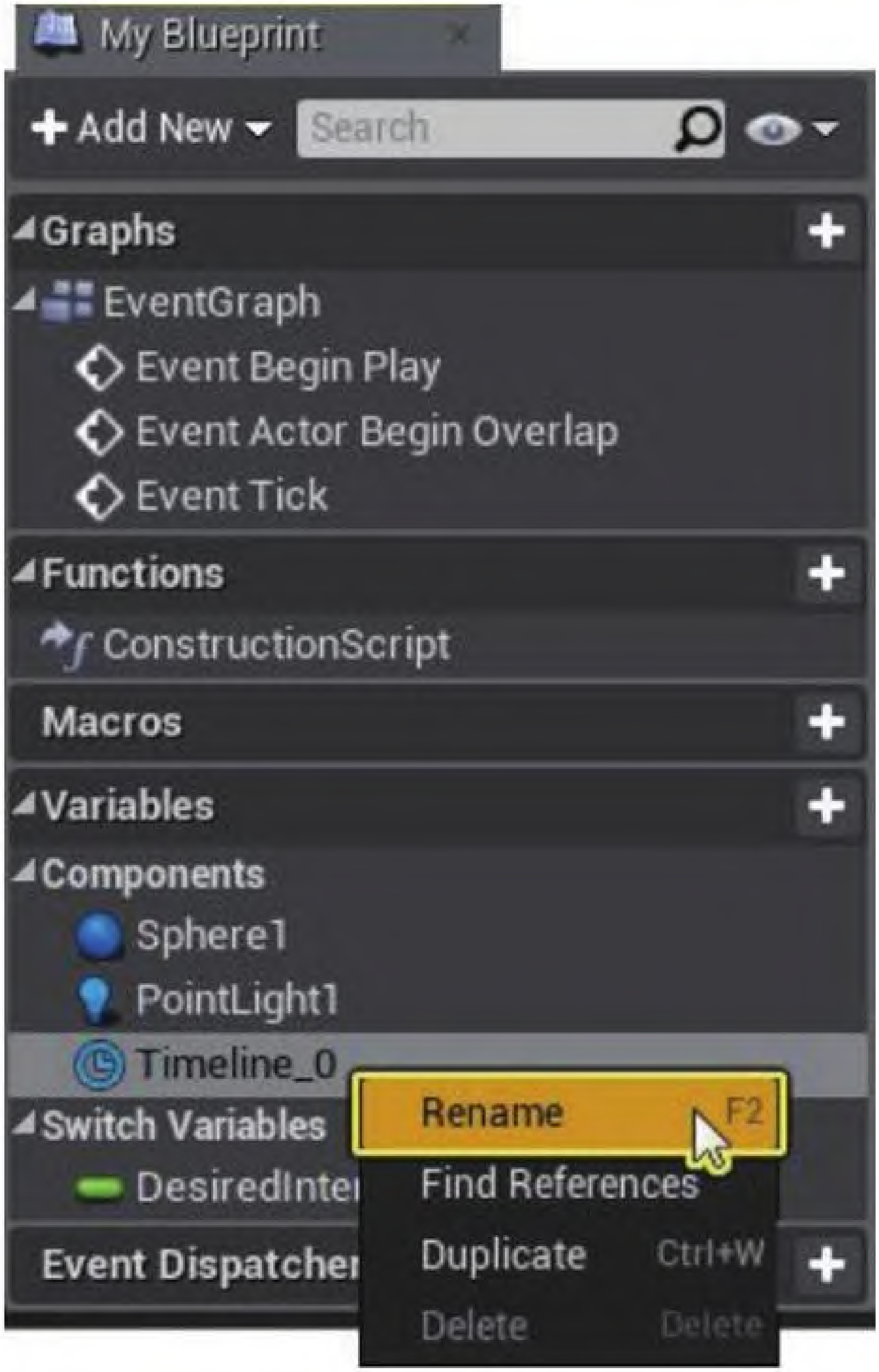



图9-142

### 5. 打开时间轴编辑器

当然，创建了时间轴后必须要对其进行设置。要打开Timeline Editor（时间轴编辑器），只需双击时间轴节点，该代码可在Graph（图表）

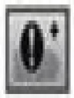
选项卡内访问，或者通过My Blueprint（我的蓝图）选项卡的列表来访问。

## 9.8.2　编辑时间轴

时间轴可以通过对图表选项卡的时间轴节点双击来编辑，或者在My Blueprint（我的蓝图）选项卡的时间轴内进行编辑。这样会在新选项卡中打开时间轴编辑器。

时间轴编辑器，如图9-143所示。

图9-143

| 按钮／选框 | 描　述 |
| --- | --- |
| | 添加新的浮点轨迹到时间轴，以对标量浮点值进行动画处理 |
| | 添加新的向量轨迹到时间轴，以对浮点向量值（例如旋转值或平移值）进行动画处理 |
|  | 添加一个事件轨迹，该轨迹会提供另一个执行输出引脚，此引脚将在轨迹的关键帧时间处被触发 |
| | 添加新的线性颜色轨迹到时间轴，以对颜色进行动画处理 |
| | 添加外部曲线到时间轴。此按钮仅在 Content Browser（内容浏览器）中选择外部曲线后才能被激活 |
| Length 5.00 | 该按钮使你能为此时间轴设置回放长度 |
| Use Last Keyframe? | 如此按钮未激活，将忽略序列的最后关键帧。这可以帮助防止动画循环时被跳过 |
| AutoPlay | 如启用该按钮，此时间轴节点无需输入即可开始，而且将在关卡一开始就开始播放 |
| Loop | 如启用该按钮，除非通过 Stop 输入引脚来停止，时间轴动画将会无限制地重复播放 |
| Replicated | 如启用，时间轴动画将跨客户端被复制 |

### 1. 添加轨迹

时间轴使用轨迹来定义单个数据的动画。可以为浮点值、向量值、颜色值和事件。轨迹可通过单击Add Track（添加轨迹）按钮之一来添加到时间轴。举例来说，单击 按钮可添加轨迹并为新轨迹输入名称。按下回车键可以为你的新浮点轨迹保存名称，如图9-144所示。

图9-144

**Step 1**　Track Name（轨迹名称）：你可以在任何时候为此区域内的轨迹输入新名称。

**Step 2**　External Curve group（外部曲线组）：使你可以从内容浏览器中选择外部曲线资源，而不用创建你自己的曲线。

**Step 3**　Track timeline（轨迹时间轴）：此轨迹的关键帧图表。你可以把关键帧放置到这里，并且你将看到作为运算结果的插值曲线。

**2.添加关键帧**

当你放置完轨迹后，你可以开始添加关键帧以定义你的动画，如图9-145所示。

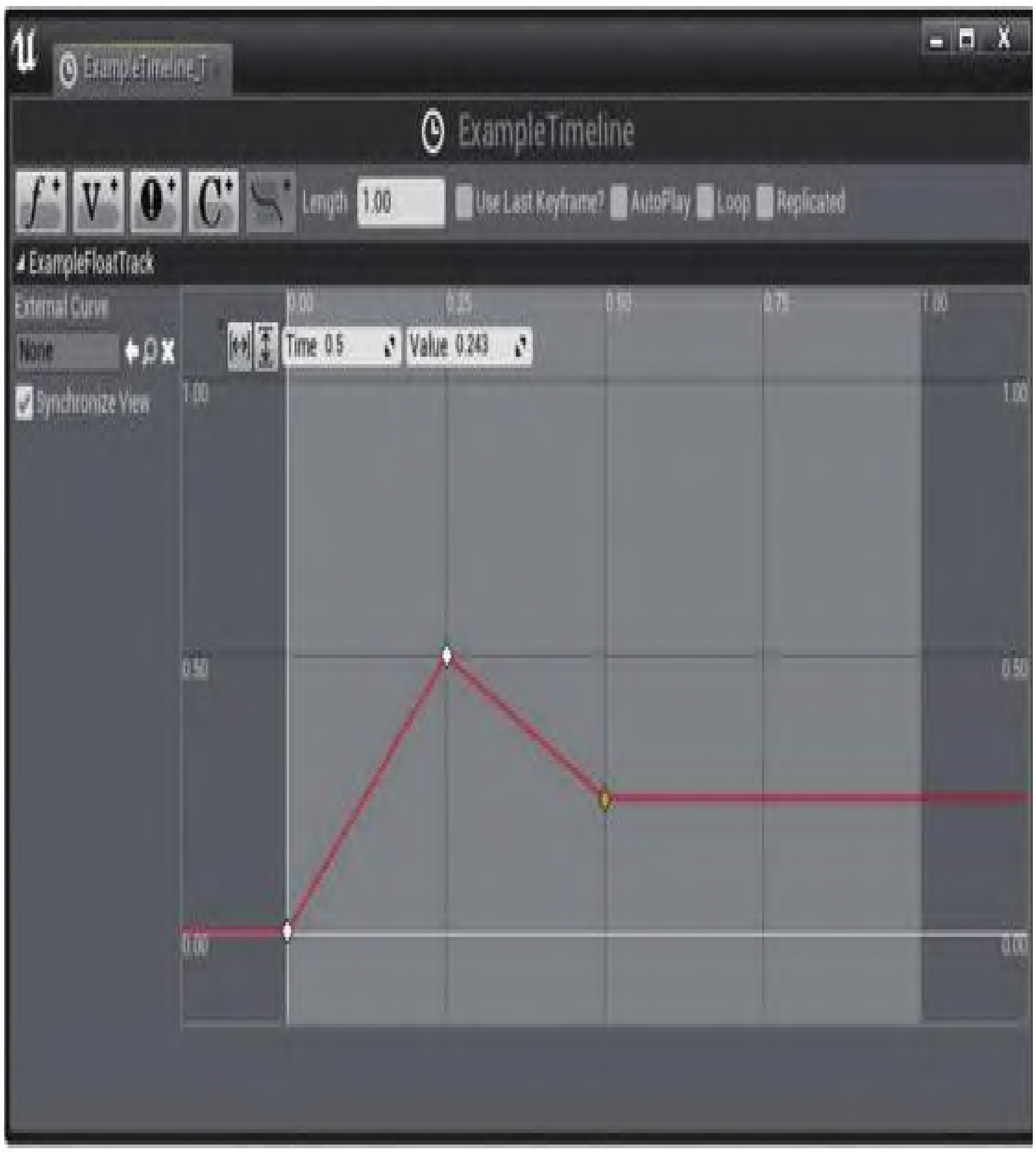



图9-145

在你完成编辑轨迹后，该轨迹的数据或事件执行将由与轨迹名称相同的数据或执行引脚来输出，如图9-146所示。

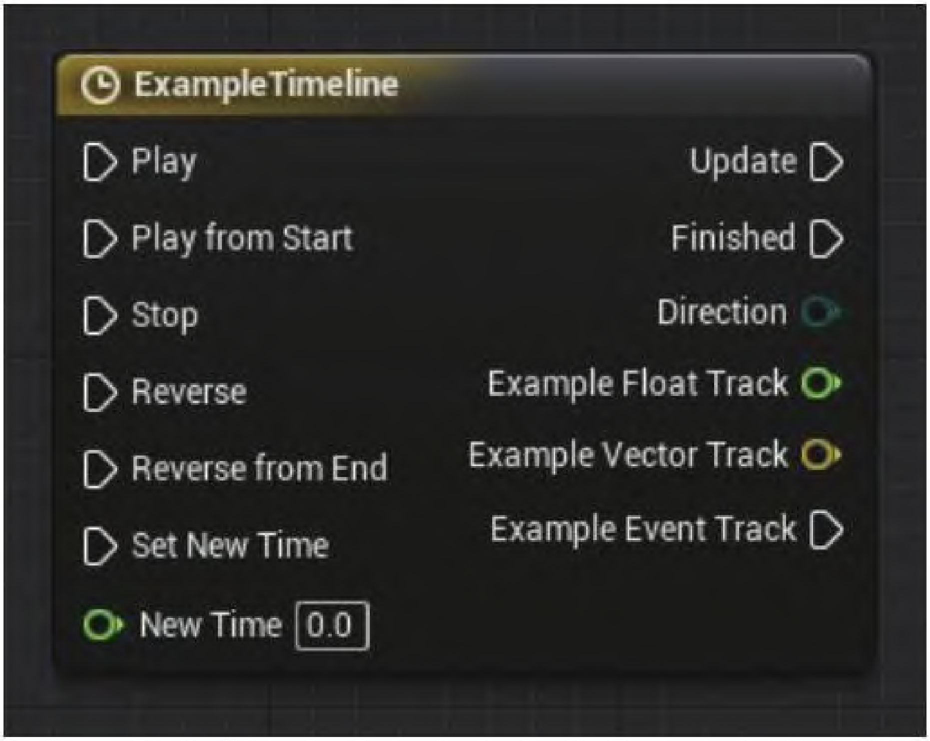



图9-146

## 9.8.3　闪光灯

**1. 概述**

这是关于时间轴的应用示例，在这里示例中，我们设置了一个可以改变颜色的闪光灯。

**2. 闪光灯**

在这个示例中，我们获得一个简单的点光源并操作它，使在玩家触发一个触发器时，该光源改变颜色，并开始闪烁，5秒后光源恢复为正常状态。

使用这个示例的好处是我们使用了时间轴中提供的三种不同类型的轨迹，如图9-147所示。

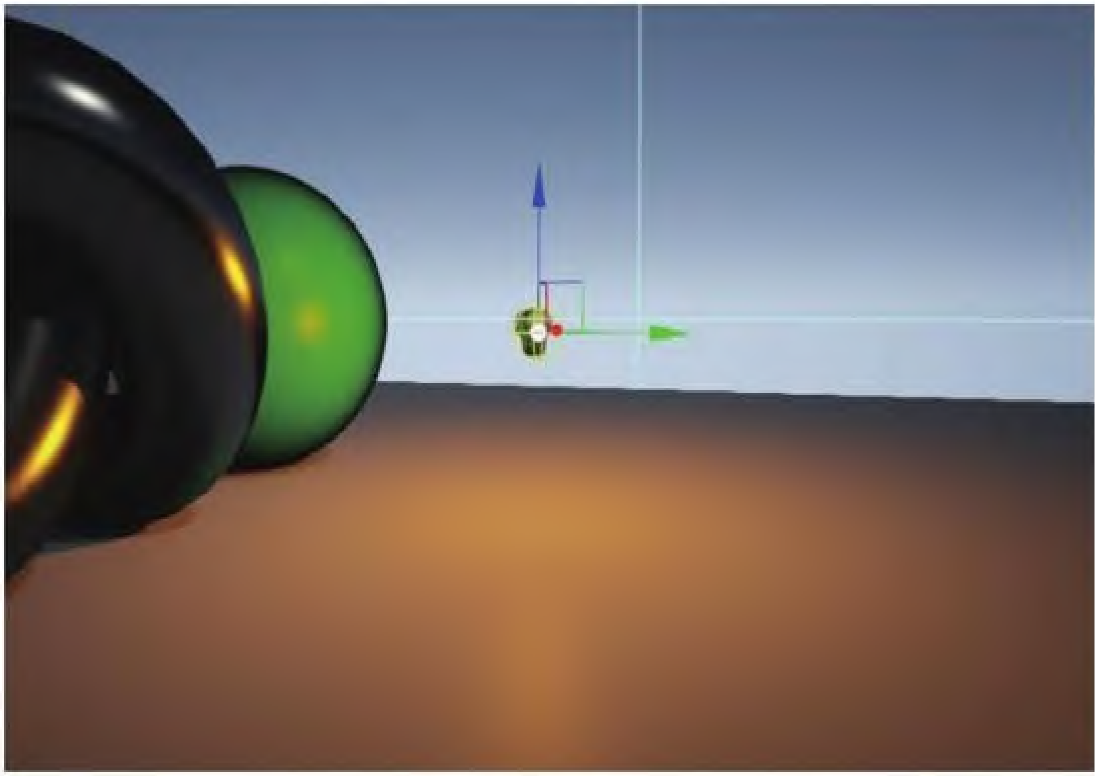


图9-147

所显示的光源将发生亮度闪烁并且会改变颜色。

设置：

时间轴提供了三种主要功能，每种功能有一种单独的轨迹。

**Step 1**　浮点型轨迹（称为Brightness（亮度））处理亮度的动画效果。该曲线是从内容浏览器中使用Flicker Curve资源添加的。

**Step 2**　向量型轨迹（称为Light Color（光源颜色））使光源的颜色产生变化。这样处理使得光源在序列开始时变为橘红色，然后在序列结尾处再变为白色。曲线插值设置为区间限定类型，这使得到下一帧之前该值仍然保持不变。

**Step 3**　事件轨迹（称为Trigger Toggle（触发器开关））在时间轴的序列播放过程中禁用我们的触发器。因为这是个事件轨迹，所以关键帧的值是无关的。任何时候，当时间轴遇到一个关键帧时，该轨迹的相关输出引脚发都会发出一个执行脉冲信号。

该网络的逻辑非常简单：因为我们总是想让这个闪光灯从一个开始就运作，所以我们的事件连入了Play from Start（从开始处运行）引脚。然而，因为我们不想在播放过程中再次触发时间轴，所以我们判断一个布尔变量的值，该变量的唯一作用是判断该时间轴是否正在播放。

该变量的默认值为false，这意味着我们总是在第一次触摸触发器时播放时间轴。所以时间轴的执行过程如下所示：

•Brightness轨迹的结果乘以20来使得光源亮度变得更高。

•光源的颜色也在下一个动作中发生了改变。

•通过使用事件轨迹，我们触发一个FlipFlop触发器，该触发器设置Is Timeline Playing变量为true。由于任何时候当某人触发了该触发器时都会判断该变量，所以一旦播放开始，我们就禁用该触发器。

•随着播放结束，事件轨迹从Toggle Trigger事件轨迹发出另一个脉冲信号，该信号再次触发FlipFlop（触发器），将我们的布尔值还原，从而重新激活触发器，如图9-148所示。

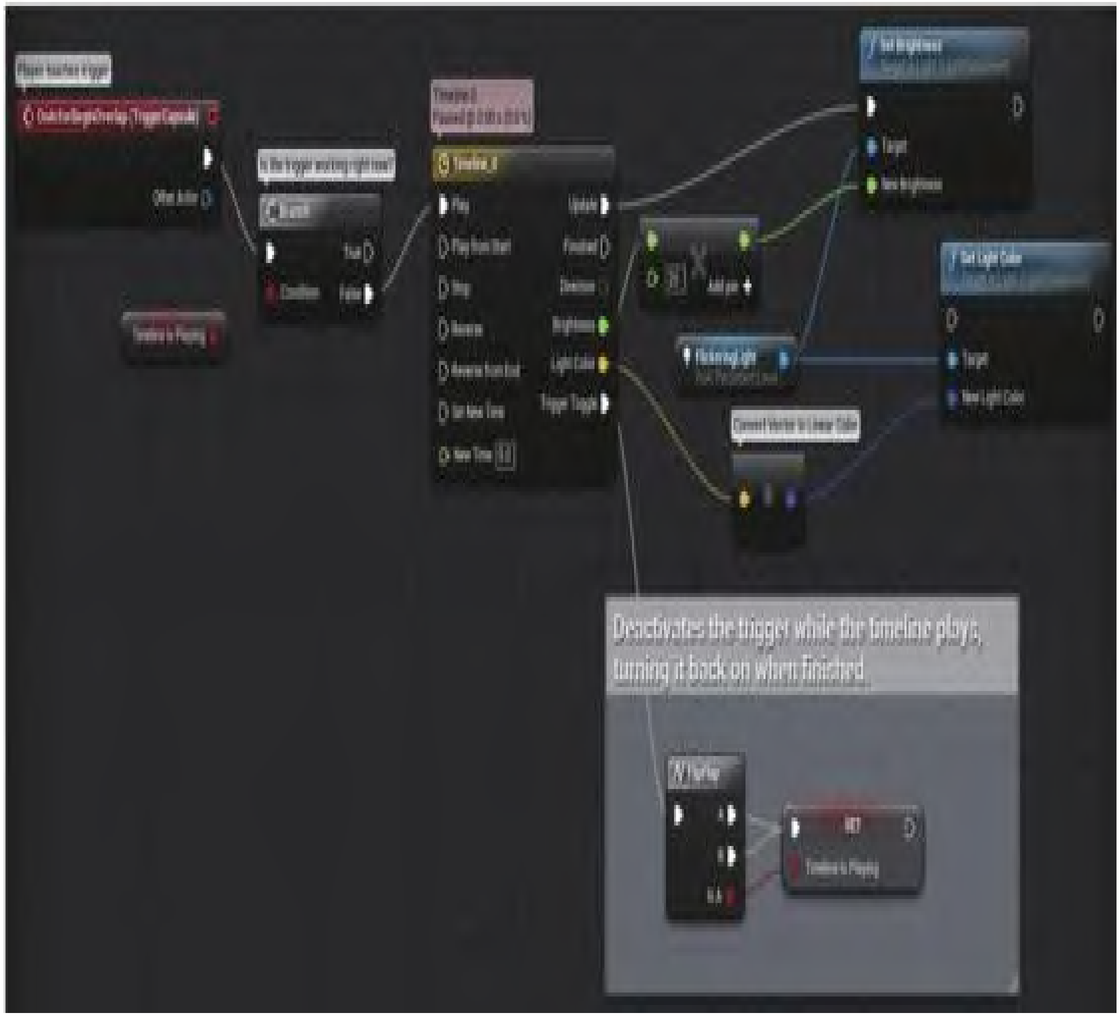



图9-148

## 9.8.4　开门

### 1. 概述

设置经典的感应开门的时间轴示例。

### 2. 关卡设置

关卡起始为简单的门体。门体两侧为单个触发空间所围绕。在玩家进入和退出此触发空间时，我们将执行时间轴来开启和关闭门体，如图9-149所示。

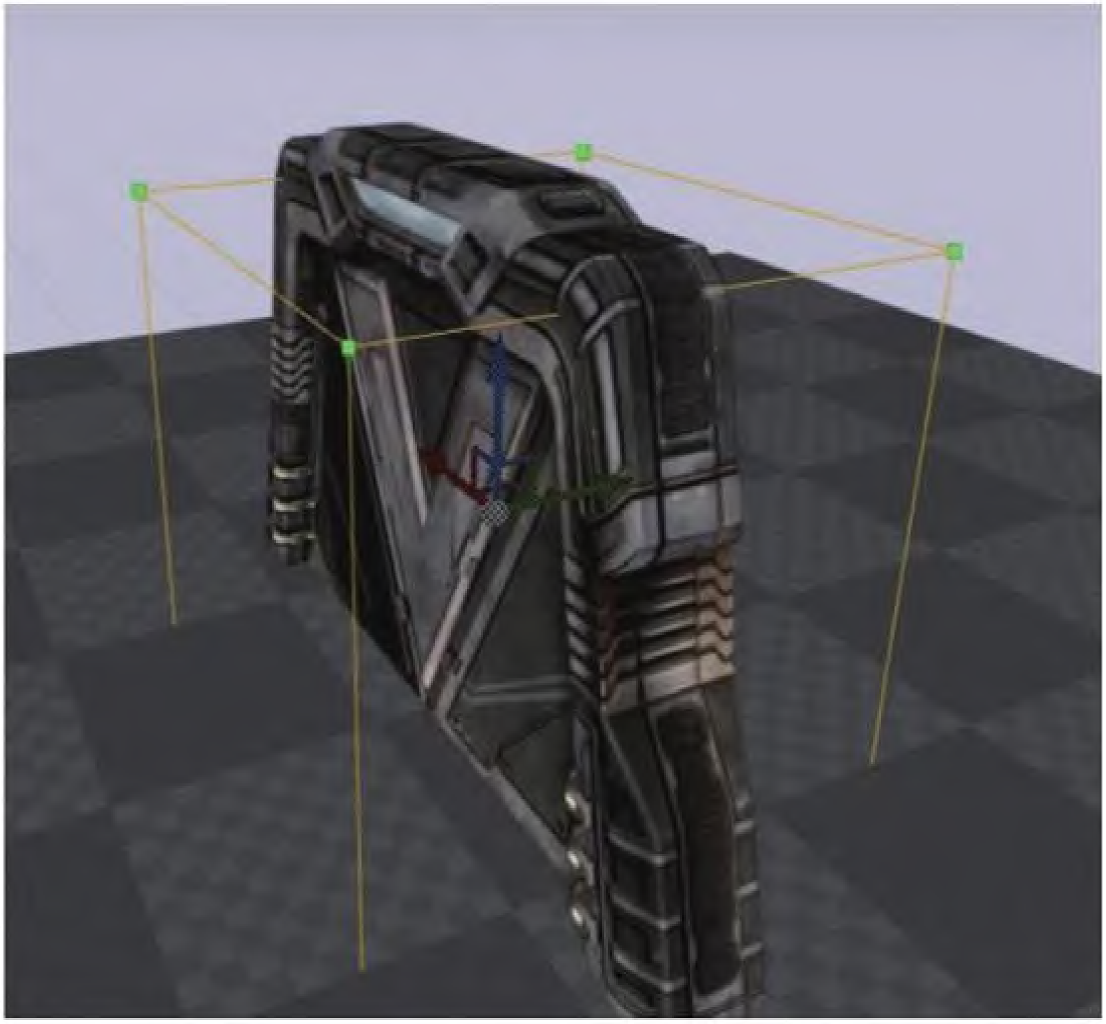


图9-149

Blueprint（蓝图）设置：

此时你可以看到驱动此行为的蓝图网络。

一开始，我们把门的当前位置存储为向量变量，其名称为“Door Initial Location”（门体初始位置）。由于和Matinee不同，时间轴不会自动做这项工作，我们需要访问该数据，这样我们可以将其添加到动作中来抬起门体。

完成后，我们就可以来设置时间轴。在这种情况下，该时间轴由2个不同的事件所调用，两者都来源于触发空间。在玩家触碰该空间时，我们播放时间轴动画，这样将会升起门体。这是通过从时间轴添加曲线值到初始位置完成的，存储在关卡的起始位置。

当玩家退出空间时，我们等待2秒然后反向播放时间轴，这样会把门重新拉下来，如图9-150所示。

图9-150

请注意本设置不考虑音效。

### 3. 时间轴设置

本例中的时间轴设置相对简单。我们已经放置了单条向量曲线。在此曲线中，我们已经通过动画处理把Z分量的值从1000提升到1360，这意味着我们在一秒内把门体向上移动360个单位。

## 9.8.5 关键帧和曲线

**1.概述**

本文介绍了如何在蓝图的时间轴编辑器中应用关键帧及曲线。

**2.应用关键帧**

每个轨迹可以具有多个关键帧，关键帧定义了一个时间和一个数值。通过在这些关键帧之间插值数据来计算在整个时间轴上任何点处的值。

**3.添加关键帧**

按住Shift键并单击灰色条，可以添加关键帧，如图9-151所示。

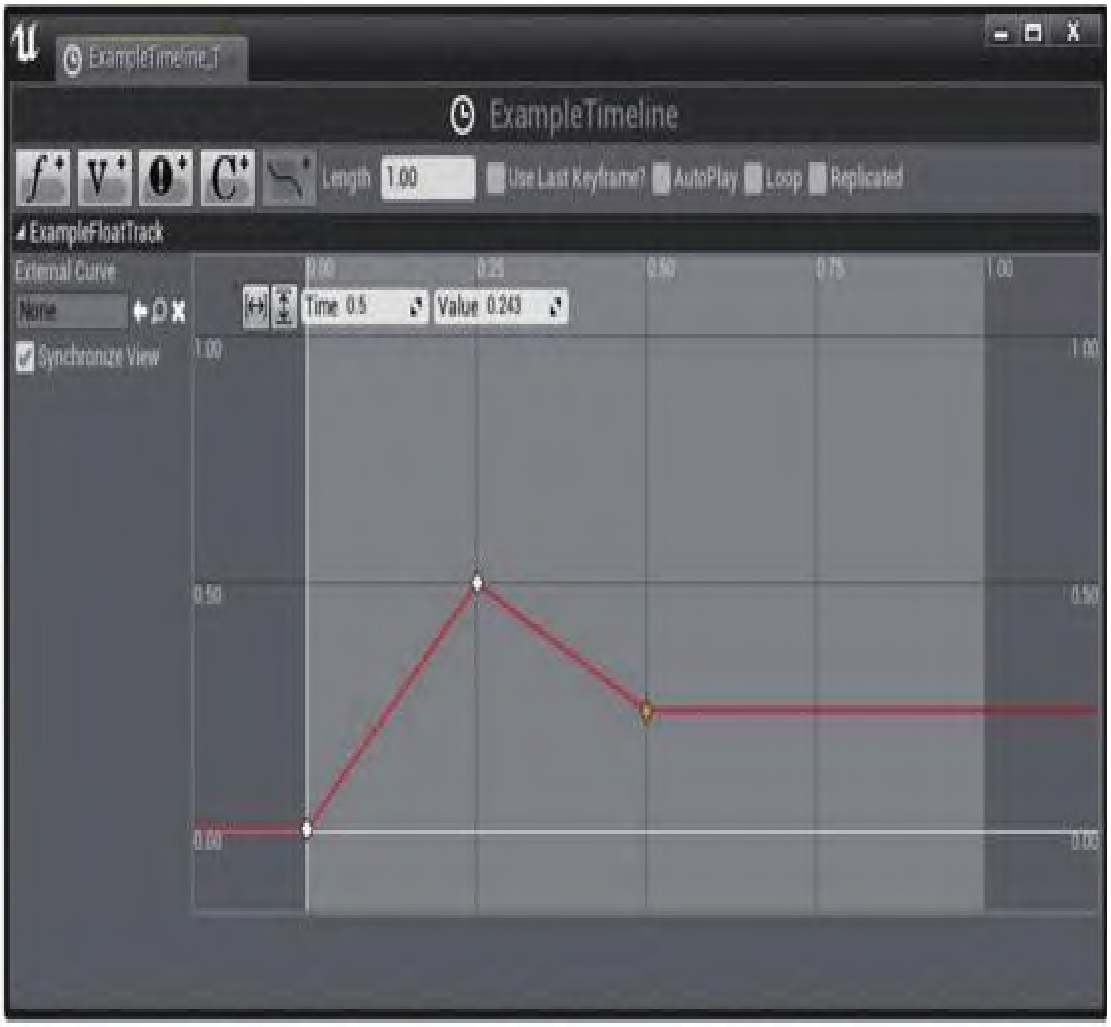



图9-151

### 4. 编辑关键帧

单击关键帧并在轨迹顶部附近的时间和数值文本框中输入数值，可以设置该关键帧的时间和数值，这两个文本输入框显示在关键帧编号旁边，如图9-152所示。

图9-152

### 5. 删除关键帧

按下键盘上的Delete（删除）键，可以删除选中的关键帧。

### 6. 移动关键帧

要想沿着时间轴移动关键帧，请选择该关键帧，然后单击并拖动它。通过使用Ctrl键，可以选中多个关键帧。水平拖动关键帧，将更新该关键帧的Time（时间），而垂直拖动关键帧，将更新Value（数值）。

### 7. 关键帧插值

通过右击一个关键帧，你可以选择给定关键帧的插值类型。同曲线插值一样，关键帧插值也具有相同的插值类型，但是关键帧插值仅影响你为其进

行插值的关键帧之间的曲线及曲线后的下一个关键帧，比如，其他所有关键帧都设置为Linear（线性），中间的关键帧（显示为橘黄色）设置为Cubic-Auto，轨迹是这样的，如图9-153所示。

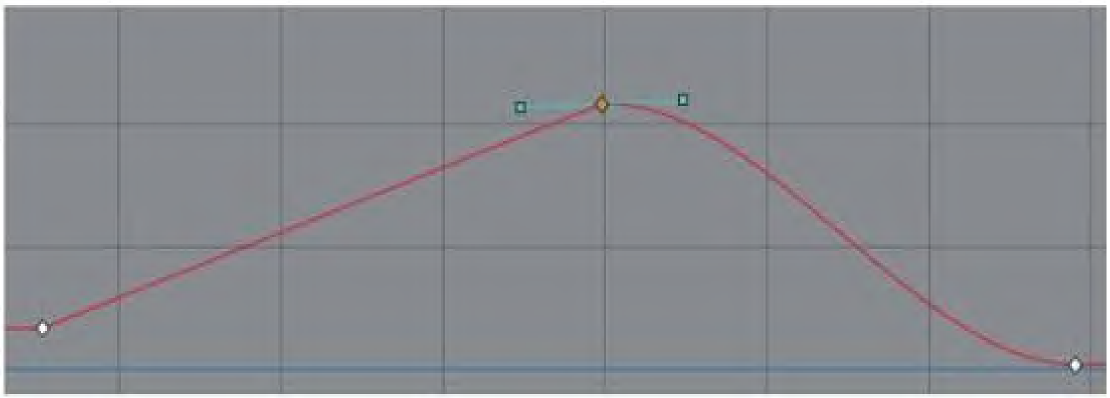


图9-153

## 8. 曲线

这部分介绍了几个关于在时间轴中应用插值曲线的重要概念。

## 9. 曲线插值

通过右击一条曲线（而不是关键帧），你可以选择给定曲线的插值类型。可用的插值类型如下。

•Linear（线性），如图9-154所示。

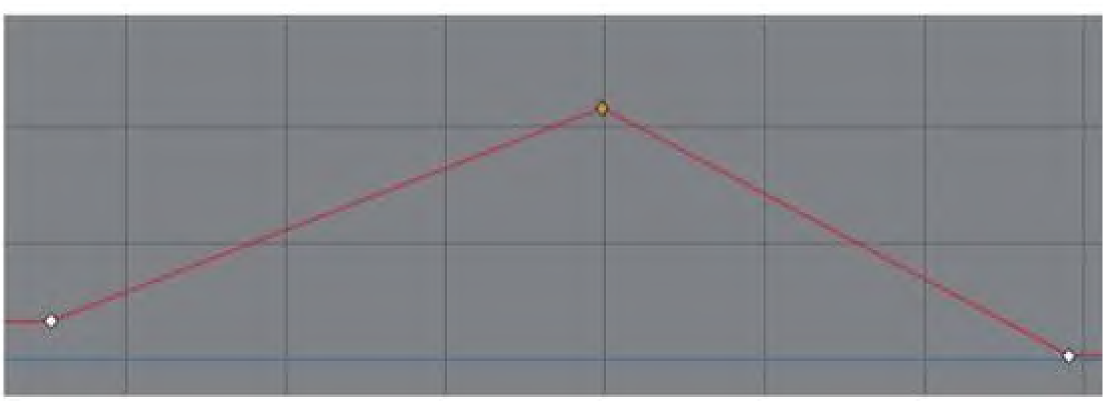


图9-154

•Clamped（区间限定），如图9-155所示。

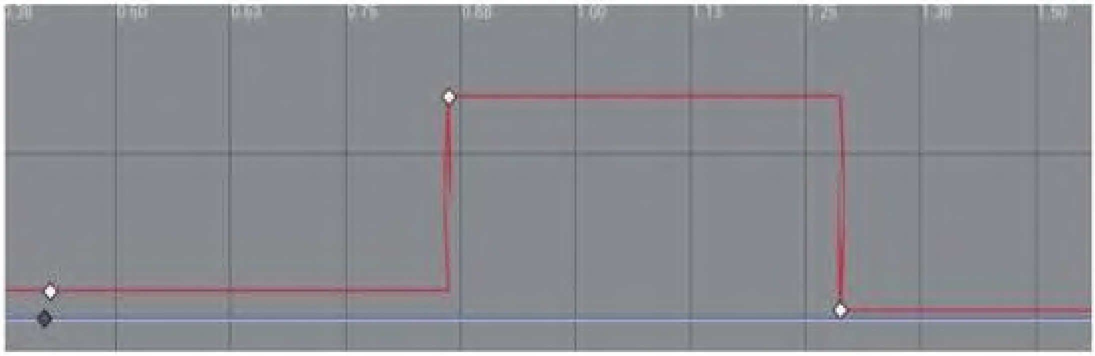


图9-155

•Cubic-Auto（立方体——自动），如图9-156所示。

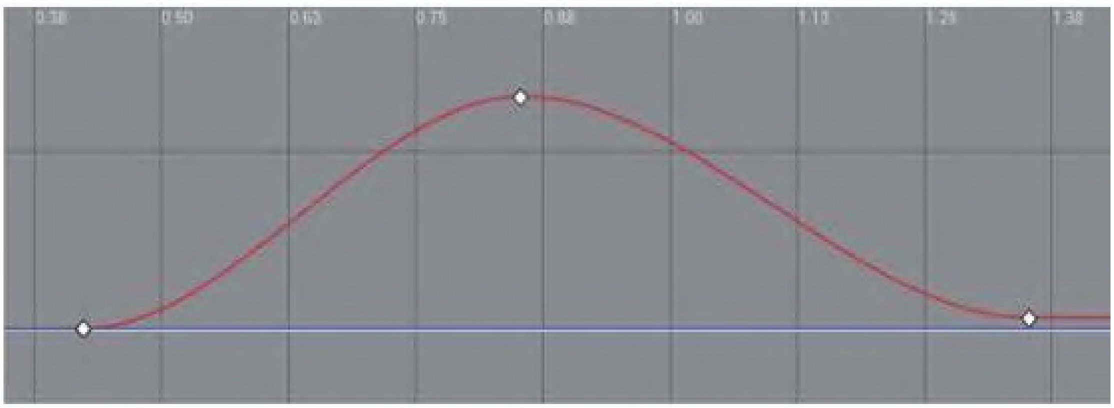


图9-156

•Cubic-Locked（立方体——锁定），如图9-157所示。

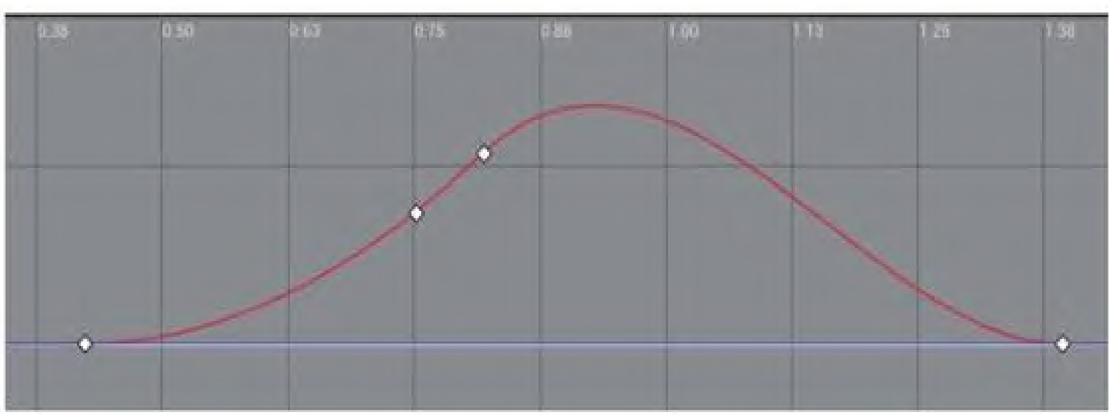


图9-157

•Cubic-Break（立方体——断开），如图9-158所示。

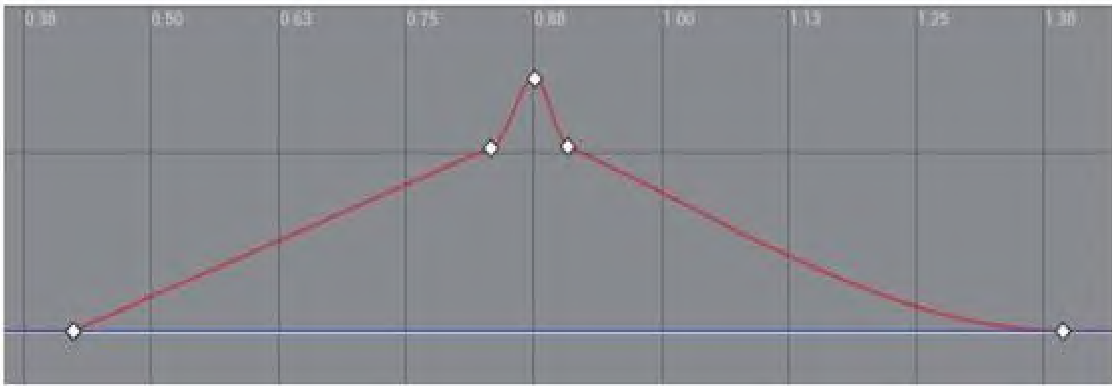


图9-158

### 10.外部曲线

在主要关键帧区域左侧的地方，你可以从内容浏览器向那个轨迹添加外部曲线资源。这和使用 按钮不同，使用 按钮将通过选中的外部曲线创建一条新轨迹。有关曲线图表见下表。

| 外部曲线图表 | 描　述 |
|---|---|
| | 为这个轨迹应用在内容浏览器中选中的曲线 |
| | 浏览内容浏览器来为该轨迹选择一条曲线 |
| | 将一条导入的外部曲线转换为内部曲线，以便可以编辑关键帧及曲线 |

你也可以右击任何曲线（至少选中一个关键帧），并选择Create External Curve（创建外部曲线）。这将弹出一个关联窗口，在这里你可以编辑保存曲线资源的路径。这样，稍后你就可以在其他时间轴上再次使用该曲线，如图9-159所示。

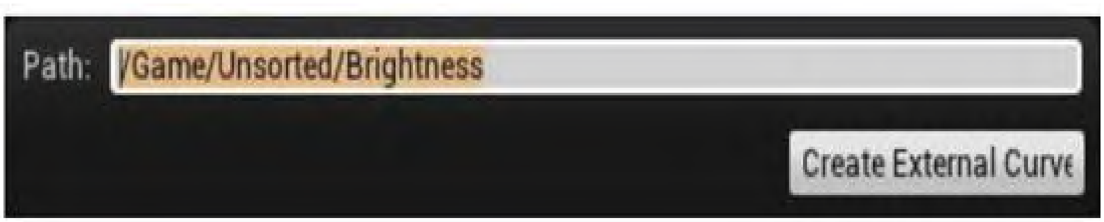


图9-159

## 9.8.6 时间轴节点

### 1. Get Play Rate（获得播放速率）

Get Play Rate（获得播放速率）节点返回输入时间轴的当前播放速率。该数值作为浮点值返回，见图9-160和下表。

图9-160

| 名　称 | 类　型 | 描　述 |
|---|---|---|
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |
| 输出 | | |
| Return Value（返回值） | 浮点型 | 输出时间轴的当前播放速率 |

### 2. Get Playback Position（获得播放位置）

Get Playback Position（获得播放位置）节点返回当前的播放位置，或输入时间轴上的当前时刻的时间。该数值作为浮点值返回，见图9-161和下表。

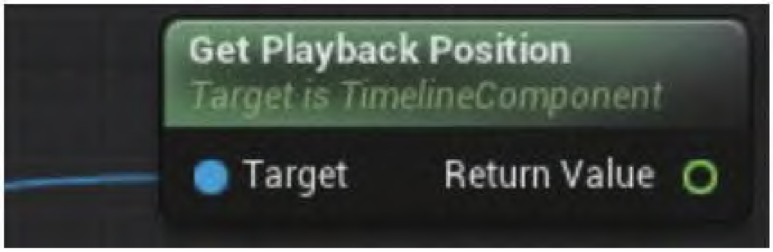



图9-161

| 名　称 | 类　型 | 描　述 |
|---|---|---|
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |
| 输出 | | |
| Return Value（返回值） | 浮点型 | 输出时间轴的当前播放位置 |

### 3. Get Timeline Length（获得时间轴长度）

Get Timeline Length（获得时间轴长度）节点返回输入时间轴的总长度，以浮点型值返回，如图9-162所示。

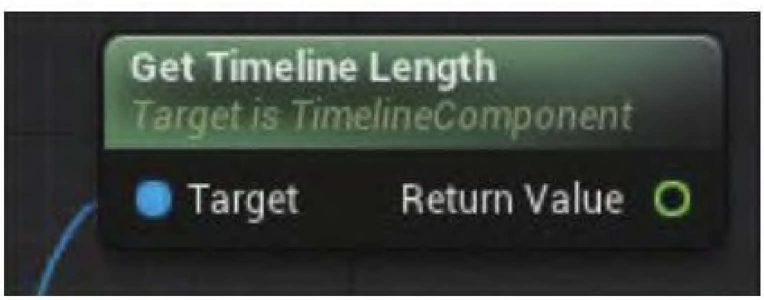



图9-162

| 名　称 | 类　型 | 描　述 |
|---|---|---|
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |
| 输出 | | |
| Return Value（返回值） | 浮点型 | 输出时间轴的总播放长度 |

#### 4. Is Looping（是否循环）

Is Looping（是否循环）节点返回一个布尔值，如果输入时间轴正在循环则返回True，如果输入时间轴没有循环，则返回False，见图9-163和下表。

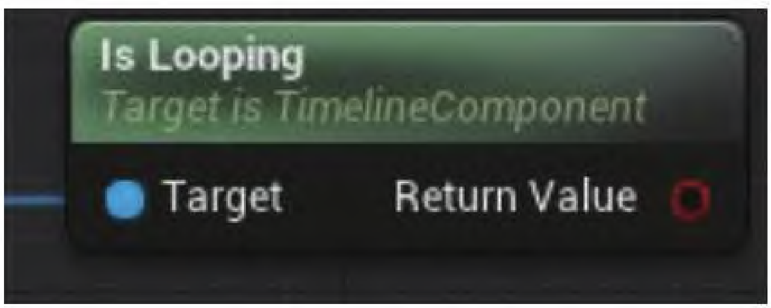



图9-163

| 名　称 | 类　型 | 描　述 |
|---|---|---|
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |
| 输出 | | |
| Return Value（返回值） | 布尔型 | 输出 Looping（循环）属性的值 |

5. Is Playing（是否正在播放）

Is Playing（是否正在播放）节点返回一个布尔值，如果输入时间轴正在播放则返回True，如果输入时间轴没有播放，则返回False，见图9-164和下表。

图9-164

| 名　称 | 类　型 | 描　述 |
|---|---|---|
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |
| 输出 | | |
| Return Value（返回值） | 布尔型 | 输出是否正在播放该时间轴 |

6. Is Reversing（是否正在反向播放）

Is Reversing（是否正在反向播放）节点返回一个布尔值，如果输入时间轴正在反向播放则返回True，如果输入时间轴没有反向播放，则返回False，见图9-165和下表。

图9-165

| 名　称 | 类　型 | 描　述 |
|---|---|---|
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |
| 输出 | | |
| Return Value（返回值） | 布尔型 | 输出是否正在反向播放该时间轴 |

### 7. Play（播放）

Play（播放）节点通知输入时间轴开始从当前播放位置处开始播放，见图9-166和下表。

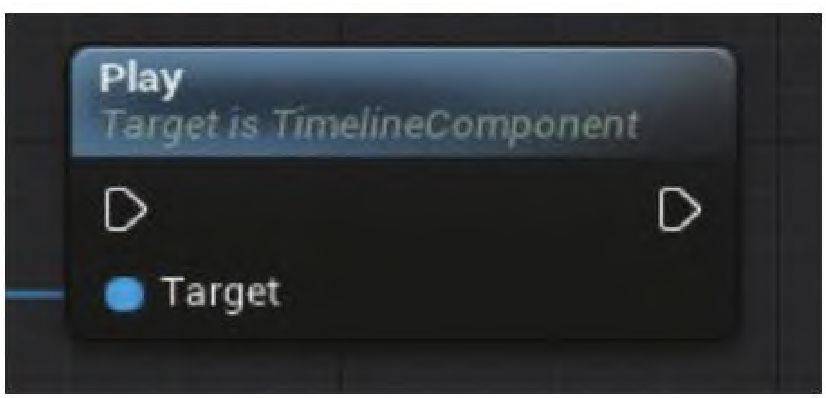



图9-166

| 名　称 | 类　型 | 描　述 |
| --- | --- | --- |
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |

### 8. Play from Start（从开始处播放）

Play from Start（从开始处播放）节点通知输入时间轴开始从时间轴的开始处开始播放，见图9-167和下表。

图9-167

| 名　称 | 类　型 | 描　述 |
| --- | --- | --- |
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |

### 9. Reverse（反向播放）

Reverse（反向播放）节点通知输入时间轴开始从当前播放位置处开始反向播放，见图9-168和下表。

图9-168

| 名　称 | 类　型 | 描　述 |
| --- | --- | --- |
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |

### 10. Reverse from End（从结尾处开始反向播放）

Reverse from End（从结尾处开始反向播放）节点通知输入时间轴开始从结尾处开始反向播放，见图9-169和下表。

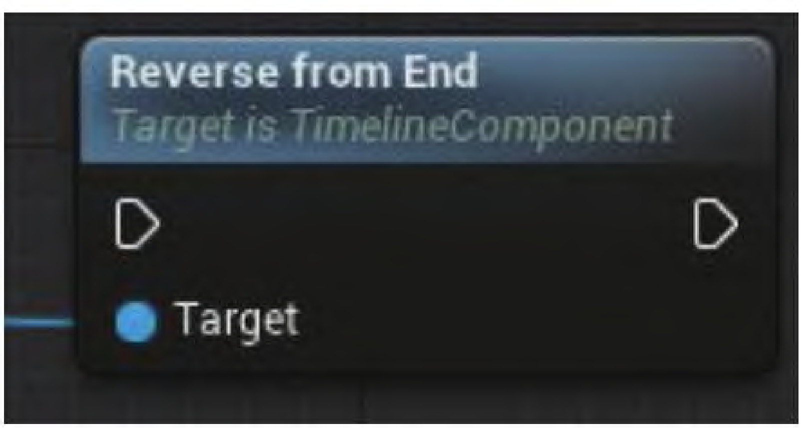



图9-169

| 名　称 | 类　型 | 描　述 |
|---|---|---|
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |

## 11. Set Looping（设置循环）

Set Looping（设置循环）节点取入一个输入时间轴和一个布尔值。它将时间轴的Looping属性设置为该布尔型值，见图9-170和下表。

图9-170

| 名　称 | 类　型 | 描　述 |
|---|---|---|
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |
| New Looping（新循环） | 布尔型 | 用于设置时间轴的循环状况的值 |

## 12. Set New Time（设置新时间）

Set New Time（设置新时间）节点取入一个输入时间轴和一个布尔值。它将时间轴的New Time属性设置为浮点型的值，见图9-171和下表。

图9-171

| 名　称 | 类　型 | 描　述 |
| --- | --- | --- |
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |
| New Time（新时间） | 浮点型 | 用于设置新的播放位置 |

13. Set Play Rate（设置播放速率）

Set New Time（设置播放速率）节点取入一个输入时间轴和一个浮点型的值。它将时间轴的Play Rate（播放速率）属性设置为该浮点型的值，见图9-172和下表。

图9-172

| 名　称 | 类　型 | 描　述 |
| --- | --- | --- |
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |
| New Rate（新速率） | 浮点型 | 用于设置新的播放速率 |

## 14. Set Playback Position（设置播放位置）

Set Playback Position（设置播放位置）节点取入一个输入时间轴、一个浮点型值和一个布尔值。它让时间轴的Playback Position（播放位置）属性跳转到浮点型值处。如果时间轴中有任何事件轨迹，那么则使用该布尔值来决定是否触发它们，见图9-173和下表。

图9-173

| 名　称 | 类　型 | 描　述 |
| --- | --- | --- |
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |
| New Position（新位置） | 浮点型 | 用于跳转到新的播放位置处 |
| Fire Events（触发事件） | 布尔型 | 用于控制是否触发跳过的任何事件（任何事件轨迹上的事件） |

15. Set Timeline Length（设置时间轴长度）

Set Timeline Length（设置时间轴长度）节点设置时间轴的总播放长度，见图9-174和下表。

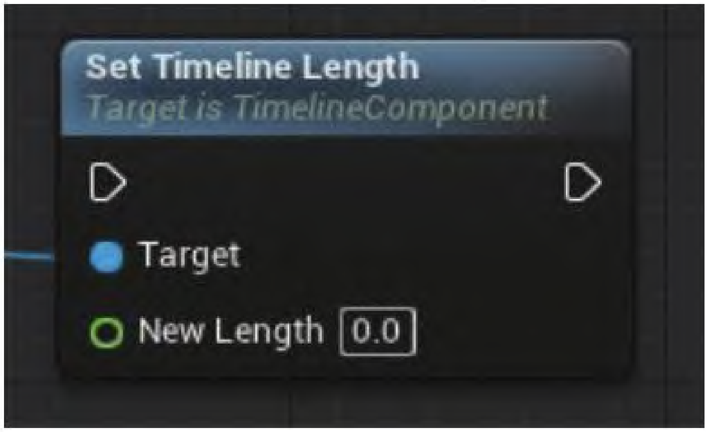



图9-174

| 名　称 | 类　型 | 描　述 |
|---|---|---|
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |
| New Length（新长度） | 浮点型 | 用于设置新的播放长度 |

### 16. Set Timeline Length Mode（设置时间轴长度模式）

Set Timeline Length Mode（设置时间轴长度模式）节点设置时间轴的Length Mode（长度模式）属性。这允许你在播放完整的时间轴和仅播放到最后一个关键帧之间进行切换，见图9-175和下表。

图9-175

| 名　称 | 类　型 | 描　述 |
| --- | --- | --- |
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |
| New Length Mode（新的长度模式） | 字节型 | 设置播放模式为"Timeline Length"（时间轴长度）或"Last Keyframe"（最后一个关键帧） |

## 17. Stop（停止）

Stop（停止）节点停止播放输入时间轴节点，见图9-176和下表。

图9-176

| 名　称 | 类　型 | 描　述 |
|---|---|---|
| 输入 | | |
| Target（目标） | 时间轴组件 | 取入一个时间轴变量 |

# 附录　蓝图术语表

B

Blueprint（蓝图）

一般术语，可以总体地指可视化的脚本系统，或者指可以定义新类或Actor类型的标准Blueprint类。

C

Custom Event（自定义事件）

通过蓝图编辑器定义的，在事件图表中可以通过节点执行的事件。

E

Event（事件）

通过代码定义的可执行的方法，没有返回值，没有任何输出参数。事件是执行的起始点，所以仅有一个输出执行引脚。事件是从原生代码中执行的。请参照Custom Events（自定义事件）。

F

Function（函数）

通过代码或蓝图编辑器定义的可执行的方法，至少具有一个输出参数或者返回一个值。函数可以是纯函数也可以是非纯函数。

Function Call（函数调用）

可以放置在事件图表中来执行一个函数的节点。

I

Impure Function（非纯函数）

可以自由修改类的状态及成员的函数。请参照纯函数和非纯函数部分来获得更多详情。

P

Private Variable（私有变量）

仅能在Blueprint Defaults（蓝图默认值）中编辑的变量，每次运行构建脚本时变量都会重置为默认值。私有变量可以用于在蓝图中跟踪状态。

Public Variable（公有变量）

可以在每个蓝图的实例中独立编辑的变量，但是在蓝图的图表中认为是常量。公有变量允许设计人员基于每个实例调整一个蓝图的外观或行为。

Pure Function（纯函数）

承诺不会修改类的状态或成员的函数。请参照纯函数和非纯函数部分来获得更多详情。